

Les fonctions

$$f(x)$$

Objectifs

- Appréhender la notion de fonctions
- Définition des fonctions
- Les fonctions en C#

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



```
*****↵
*           *↵
*           *↵
*           *↵
*****↵
```

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10
ç

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



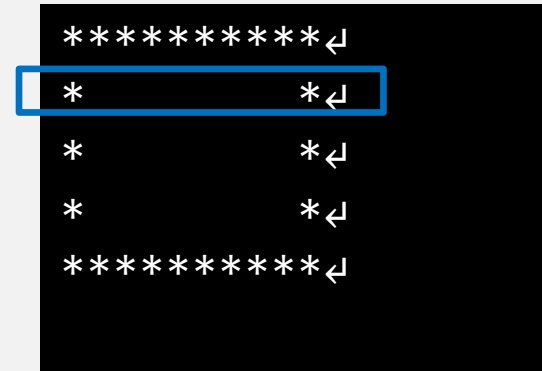
```
*****↵
*      *↵
*      *↵
*      *↵
*****↵
```

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



```
*****↵
*      *↵
*      *↵
*      *↵
*****↵
```

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }

    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



```
*****↵
*           *↵
*           *↵
*           *↵
*****↵
```

Problème


- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1 {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }

    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```



The diagram illustrates the output of the code: a 5x10 grid of asterisks. The first and last rows are highlighted with red boxes, representing the first and last iterations of the 'hauteur' loop. The three middle rows are highlighted with a green box, representing the iterations of the 'hauteur' loop where the 'ligne' loop is executed. Within the green box, the first and last columns are highlighted with blue boxes, representing the iterations of the 'colonne' loop for the first and last lines of the middle section. This visualizes the nested loop structure and the repetition of code for each row and column.

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {
    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1 {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }

    pour entier colonne de 1 à largeur {
        écrire('*');
    }
    écrireNL();
}
```

```
*****↵
*           *↵
*           *↵
*           *↵
*****↵
```

Ces deux bouts de pseudocode se répètent

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {

    // Afficher 'largeur' étoiles

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }

    // Afficher 'largeur' étoiles

}
```

```
*****↵
*           *↵
*           *↵
*           *↵
*****↵
```

Fonction simple

- Bloc de code
 - Possède un nom
 - Toujours avec des parenthèses (On verra pourquoi ...)
 - Écrit une seule fois
 - Peut être utilisé une ou plusieurs fois

```
aucun AfficherLigne10Etoiles()  
{  
    pour entier colonne de 1 à 10 {  
        écrire('*');  
    }  
    écrireNL();  
}
```

```
*****↵
```

Fonction simple

- Utilisation :
 - On spécifie le nom de la fonction suivi de parenthèses :

```
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();
```

```
*****↵  
*****↵  
*****↵  
*****↵
```

Fonction avec paramètres – Exemple 1

- Une fonction peut prendre des paramètres
 - Un paramètre a un type et un nom : il est similaire à la déclaration d'une variable
 - Chaque nom de paramètre doit être préfixé par "p_"

```
aucun AfficherLigneEtoiles(entier p_largeur)
{
    pour entier colonne de 1 à p_largeur {
        écrire('*');
    }
    écrireNL();
}
```

[...]↵

Fonction avec paramètres – Exemple 1

- Utilisation :
 - On spécifie la valeur des paramètres en les plaçant dans les parenthèses de l'appel

```
AfficherLigneEtoiles(4);  
AfficherLigneEtoiles(3);  
AfficherLigneEtoiles(2);  
AfficherLigneEtoiles(1);
```

```
****  
***  
**  
*
```

Problème

- Comment éviter de répéter du code ?

```
entier hauteur = lire(); // 5
entier largeur = lire(); // 10

si (hauteur >= 2 et largeur >= 2) alors {

    AfficherLigneEtoiles(largeur);

    pour entier ligne de 2 à hauteur - 1 {
        écrire('*');
        pour entier colonne de 2 à largeur - 1
        {
            écrire('_');
        }
        écrire('*');
        écrireNL();
    }

    AfficherLigneEtoiles(largeur);

}
```



```
*****↵
*           *↵
*           *↵
*           *↵
*****↵
```

Fonction avec paramètres – Exemple 2

- Une fonction peut prendre des paramètres
 - Les paramètres sont séparés par des ","

```
aucun AfficherLigneMotif(entier p_largeur, caractère p_motif)
{
    pour entier colonne de 1 à p_largeur {
        écrire(p_motif);
    }
    écrireNL();
}
```

#[...]#↵

Fonction avec paramètres – Exemple 2

- Utilisation :
 - On spécifie la valeur des paramètres en les plaçant dans les parenthèses de l'appel
 - Les valeurs sont séparées par des ","

```
AfficherLigneMotif(4, '&');  
AfficherLigneMotif(3, '#');  
AfficherLigneMotif(2, '@');  
AfficherLigneMotif(1, '#');
```

```
&&&&↵  
###↵  
@@↵  
#↵
```

Fonction avec un retour – Exemple 3

- Une fonction peut renvoyer une valeur.
 - Le type de retour doit être spécifié
 - Pour renvoyer une valeur on utilise l'instruction "renvoyer"
 - Une fois cette instruction exécutée, la fonction est quittée

```
entier CalculerMinimum(entier p_premiereValeur, entier p_deuxiemeValeur)
{
    entier resultat = p_premiereValeur;

    si (p_deuxiemeValeur < resultat) alors {
        resultat = p_deuxiemeValeur;
    }

    renvoyer resultat;
}
```

Fonction avec un retour – Exemple 3

- Utilisation :
 - La valeur renvoyée doit être utilisée

```
entier valeur1 = 13;  
entier valeur2 = 42;  
  
entier minimum = CalculerMinimum(valeur1, valeur2);  
  
écrireNL(minimum);
```

13↵

« Fonctions » (Méthodes statiques)

- À partir de maintenant, déclarez vos fonctions dans la classe / le fichier « *Fonctions.cs* »
- Mettre le mot clef ***public*** devant le nom de la classe
- Pour rappel, pour déclarer une fonction vous allez devoir la déclarer comme étant « ***public static*** »
- L'appel des fonctions contenues dans ce fichier se fait en préfixant le nom des fonctions avec le nom de la classe et d'un point « *Fonctions.* »

Fonction simple – Et C# ?

```
aucun AfficherLigne10Etoiles()  
{  
    pour entier colonne de 1 à 10 {  
        écrire('*');  
    }  
    écrireNL();  
}
```

```
public static void AfficherLigne10Etoiles()  
{  
    for (int colonne = 1; colonne <= 10; colonne++) {  
        Console.Out.Write('*');  
    }  
    Console.Out.WriteLine();  
}
```

```
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();
```

```
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();  
AfficherLigne10Etoiles();
```

Fonction avec paramètres – Exemple 1 – Et C# ?

```
aucun AfficherLigneEtoiles(entier p_largeur)
{
    pour entier colonne de 1 à p_largeur {
        écrire('*');
    }
    écrireNL();
}
```

```
AfficherLigneEtoiles(4);
AfficherLigneEtoiles(3);
AfficherLigneEtoiles(2);
AfficherLigneEtoiles(1);
```

```
public static void AfficherLigneEtoiles(int p_largeur)
{
    for (int colonne = 1; colonne <= p_largeur; colonne++) {
        Console.Out.Write('*');
    }
    Console.Out.WriteLine();
}
```

```
AfficherLigneEtoiles(4);
AfficherLigneEtoiles(3);
AfficherLigneEtoiles(2);
AfficherLigneEtoiles(1);
```

Fonction avec paramètres – Exemple 2 – Et C# ?

```
aucun AfficherLigneMotif(entier p_largeur,  
                           caractère p_motif)  
{  
    pour entier colonne de 1 à p_largeur {  
        écrire(p_motif);  
    }  
    écrireNL();  
}
```

```
public static void AfficherLigneMotif(int p_largeur,  
                                       char p_motif)  
{  
    for (int colonne = 1; colonne <= p_largeur; colonne++) {  
        Console.Out.Write(p_motif);  
    }  
    Console.Out.WriteLine();  
}
```

```
AfficherLigneMotif(4, '&');  
AfficherLigneMotif(3, '#');  
AfficherLigneMotif(2, '@');  
AfficherLigneMotif(1, '#');
```

```
AfficherLigneMotif(4, '&');  
AfficherLigneMotif(3, '#');  
AfficherLigneMotif(2, '@');  
AfficherLigneMotif(1, '#');
```

Fonction valeur de retour – Exemple 3 – Et C# ?

```
entier CalculerMinimum(entier p_premiereValeur,  
                       entier p_deuxiemeValeur)  
{  
    entier resultat = p_premiereValeur;  
  
    si (p_deuxiemeValeur < resultat) alors {  
        resultat = p_deuxiemeValeur;  
    }  
  
    renvoyer resultat;  
}
```

```
public static int CalculerMinimum(int p_premiereValeur,  
                                  int p_deuxiemeValeur)  
{  
    int resultat = p_premiereValeur;  
  
    if (p_deuxiemeValeur < resultat) {  
        resultat = p_deuxiemeValeur;  
    }  
  
    return resultat;  
}
```


Validation des paramètres

- Pour valider les préconditions, nous allons vérifier leurs conditions inverses et lever une exception si la précondition n'est pas respectée

```
public static void AfficherLigneMotif(int p_largeur,
                                     char p_motif)
{
    if (p_largeur < 1)
    {
        throw new ArgumentException("La largeur ne doit pas être
inférieure à 1", "p_largeur");
    }

    for (int colonne = 1; colonne <= p_largeur; colonne++) {
        Console.Out.Write(p_motif);
    }
    Console.Out.WriteLine();
}
```

if (p_largeur < 1) : inverse d'une précondition
throw new ArgumentException(...) : levée d'une exception