Fonction

Les fonctions sont des blocs de code qui peuvent être appelé n'importe quand dans le code.

Une fonction est une série d'instructions qui permet de réaliser une tâche précise.

Elle peut recevoir un ou des `parametres` en entrée et peut renvoyer une variable en sortie, le `retour`.

Visualiser une fonction

Une fonction peut être vue comme une usine avec un tapis roulant d'entrée (`parametres`) et un tapis roulant de sortie (`retour`).

On donne un ou des matières premières à l'usine en entrée (`parametres`), et l'usine nous ressortira un produit fini (`retour`).

Organiser son code

Une fonction permet d'organiser son code pour plus de clarté, de séparer les tâches en bloc précis.

```
void Main()
{
// code principal
}
void fonction1()
{
// code de la fonction 1
}
```

Scope de fonction

Les variables créent dans une fonction ne sont pas visibles dans les autres fonctions.
La fonction va déclarer son scope, donc son champ d'action.

```
void Main()
{
int a = 1;
fonction1();
Console.WriteLine("var in main " + a); // affiche 1
}
void fonction1()
{
int a = 2;
Console.WriteLine("var in function " + a);
}
```

Paramètres passés

Les paramètres sont l'entrée d'une fonction, ils permettent de passer des variables à la fonction.

```
void main()
{
int a = 3;
int b = 8;
function1(a, b);
}
void fonction1(int a, int b)
{
Console.WriteLine(a + b); // affiche 11;
}
```

Paramètres passés

Le nom des paramètres n'est pas obligatoirement identique à celui passé dans la fonction.

Ils seront, dans la fonction, génériques pour pouvoir être clair.

Le Type par contre doit être le même.

```
void main()
{
int a = 3;

square(a);
print(a);
}

void square(int number_to_quare)
{
Console.WriteLine(a * a); // affichera 9;
}

void print(string str) // error, parametre string, parametre passe int
{
Console.WriteLine(str); // affichera la string;
}
```

Paramètre par default

Les paramètres peuvent avoir une valeur par default, si il n'est pas passer a la fonction.

```
void main()
{
int a = 3;
int b = 8;
function1(a, b);  // affichera 11
function1(a);   // affichera 5
}
void function1(int a, int b = 2)
{
Console.WriteLine(a + b);
}
```

Retour fonction

Une fonction peut renvoyer un résultat, ce résultat est un `type` de variable.
Ce `type` est place devant le nom de la fonction, et celle ci ne pourra renvoyer que ce `type`.

On utilisera le mot clef `return` suivit de la variable que l'on voudra retourner.

```
return_type                fonction_name( parametre, ...) {
return variable_name ;
}
```

Si une fonction ne renvoie rien, on dit qu'elle est de type `void`.

Exemple :

```
void main()
{
int a = 3;
int b = 8;
Console.WriteLine(square(a)); // affichera 9
Console.WriteLine(square(b)); // affichera 64

}
int square(int x)
{
return (x * x);
}
```

Associer le retour d'une fonction

On peut associer le retour d'une fonction à une variable.

```
void main() {
        int a = 3;
        int b = 8;
        int x = square(a);
        int y = square(b);
        Console.WriteLine(x); // affichera 9
        Console.WriteLine(y); // affichera 64
}

int square(int x) {
        return (x * x);
}
```

Exercice

- 📌 créer une fonction qui prend en paramètre un tableau d'entier, et qui renvoie le plus grand élément du tableau.
- 📌 créer une fonction qui prend en paramètre un tableau d'entier, et qui renvoie la somme de tous les éléments du tableau.

Référence

Qu'est ce qu'une référence?

Une référence permet d'accéder à la variable et la modifier en dehors de son scope dans des fonctions.

Utiliser la référence

La référence est utile dans des fonctions qui ont pour but de modifier les variables en paramètre.

En C# certaines variable comme les `tableaux` sont de type référence de base.

Les variables classiques, `int`, `float` , `string` ne le sont pas.

```
main();

void function1(int[] arr) {
arr[0] = 100;
};
void function2(int num) {
num = 10;
};

void main() {

int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int num = 18;

Console.WriteLine("before go to fonction1 " + arr[0]); // affiche 1
Console.WriteLine("before go to fonction2 " + num);  // affiche 18

function1(arr);
function2(num);


Console.WriteLine("after fonction1 "+ arr[0]); // affiche 100
Console.WriteLine("after fonction2 "+ num); // afficher 18

};
```

Comment utiliser la reference

Mot clef `ref` devant le type de la variable dans les paramètre de la fonction.

```
main();

void function1(ref int num) { // mot clef ref
num = 1000;
};

void main() {

int num = 18;

Console.WriteLine("before go to fonction2 " + num); // affichera 18
function1(ref num); // mot clef ref
Console.WriteLine("after fonction2 "+ num); // affichera 1000

};
```

Exercice avec les references

Écrire une fonction `is_Billi` qui prend en paramètre une référence sur string, renvoie un booléen.
La fonction vérifie si l'argument est égale à `Billi` et renverra `true` si vrai.
Sinon modifier l'argument pour le transforme en `Billi` et renvoyer false.

Librairie

Une librairie est un ensemble de fonction déjà écrite, utilisable dans notre programme, réunit dans une classe.

Tout au long de ce cour nous avons utiliser `Console.WriteLine`, Console étant un classe avec des fonctions comme WriteLine.

Random

La classe `Random` permet de générer des nombres aléatoires.

```
Random rnd = new Random();
int x = rnd.Next(0, 100); // x est un nombre aleatoire entre 0 et 100
```

Dictionnaire

La classe `Dictionary` permet de créer un tableau de `clef / valeur`.
Les avantages sont :

- 📌 les méthodes de base a dispositions.
- 📌 Le faites que le "tableau" soit extensible.
- 📌 La recherche des éléments rapide.

Déclaration

Un dictionnaire ce déclare avec 2 types;
Un type pour la `clef`, un pour la `valeur`.

```
Dictionary<int , string> dict = new Dictionary<string, int>(); // int --> clef , string --> valeur
```

Clef Valeur

Un dictionnaire associe une clef a une valeur.

Une clef est unique.
Pour rechercher une valeur, nous utiliserons sa clef comme index.

```
Dictionary<int, string> dictionnaire = new Dictionary<int, string>();
dictionnaire.Add(1, "un");
dictionnaire.Add(2, "deux");
dictionnaire.Add(3, "trois");

Console.WriteLine(dictionnaire[1]); // affichera un
Console.WriteLine(dictionnaire[2]); // affichera deux
Console.WriteLine(dictionnaire[3]); // affichera trois
```

Fonction recherche

La fonction `ContainsKey` permet de rechercher si une clef existe dans le dictionnaire.

```
Dictionary<int, string> dictionnaire = new Dictionary<int, string>();
dictionnaire.Add(1, "un");
dictionnaire.Add(2, "deux");
dictionnaire.Add(3, "trois");

if (dictionnaire.ContainsKey(1)) {
Console.WriteLine("La clef 1 existe dans le dictionnaire");
}
```

Méthode Dictionary

- `Add` : permet d'ajouter une clef / valeur
- `Remove` : permet de supprimer une clef / valeur
- `Clear` : permet de vider le dictionnaire
- `ContainsKey` : permet de vérifier si une clef existe
- `ContainsValue` : permet de vérifier si une valeur existe