Fonction

Fonction

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

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

Fonction

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 affiche tout les elements du tableau.
  Nous la nommerons `printTab`.
  Elle aura pour prototype : `void printTab(int[] tab)`
- 📌 Créer une fonction qui prend en paramètre un tableau d'entier, et qui renvoie le plus grand élément du tableau.
  Nous la nommerons `maxTab`.
  Elle aura pour prototype : `int maxTab(int[] tab)`

Exercice

📌 Créer une fonction qui prend en paramètre un tableau d'entier, et qui renvoie la somme de tous les éléments du tableau.
Nous la nommerons `sumTab`.
Elle aura pour prototype : `int sumTab(int[] tab)`

Exercice Corrige

```
int [] tab = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

printTab(tab);

void printTab(int[] tab) {
        for (int i = 0; i < tab.Length; i++) {
                Console.WriteLine(tab[i]);
        }
}
```

Exercice Corrige

```
int [] tab = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int max = maxTab(tab);


int maxTab(int[] tab) {
        int max = tab[0];
        for (int i = 1; i < tab.Length; i++) {
                if (tab[i] > max) {
                        max = tab[i];
                }
        }
        return max;
}
```

Exercice Corrige

```
int [] tab = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int sum = sumTab(tab);

int sumTab(int[] tab) {
        int sum = 0;
        for (int i = 0; i < tab.Length; i++) {
                sum += tab[i];
        }
        return sum;
}
```

Exercice

- 📌 Créer une fonction qui remplit un tableau d'entier avec des nombres aléatoires, comprit dans entre `a` et `b`, parametre de la fonction.
  Elle aura pour prototype :
  ```
  void fillTab(int[] tab, int a, int b)
  ```
- 📌 Modifier la fonction `fillTab` pour qu'elle prenne par default `a = 0` et `b = 100` si les parametres ne sont pas passer.

Exercice Corrige

```
void printTab(int[] tab)
{
    for (int i = 0; i < tab.Length; i++)
    {
        Console.Write(tab[i] + " ");
    }
    Console.WriteLine();
}

void fillTab(int[] tab, int a = 0, int b = 100)
{
    var random = new Random();
    for (int i = 0; i < tab.Length; i++)
    {
        tab[i] = random.Next(a, b);
    }
}

int tabSize = Convert.ToInt32(Console.ReadLine());
int[] tab = new int[tabSize];

fillTab(tab, 100, 1000);
printTab(tab);
```