

Paradigmes de programmation

Il existe plusieurs manières d'aborder la communication avec les machines au travers de programmes contenant des instructions, ce qu'on nomme *paradigme*.

Impératif ou Déclaratif

On distingue 2 ensembles principaux:

1. Impératif
2. Déclaratif

Impératif

L'idée est de donner les ordres de manière séquentielle à l'ordinateur qui exécutera les instructions. Ce modèle comprend notamment les paradigmes suivants:

- **Procédural**
- **Orienté objet**

Exemple

Pour afficher 1,2,3,4,5, on pourrait avoir le pseudo-code suivant:

```
Afficher 1
Afficher 2

DÉCLARER i
POUR i de 3 à 5 FAIRE
    AFFICHER i
FIN POUR
```

Ce code va imposer une suite d'opérations à faire par l'ordinateur dans le but de **construire** un résultat.

Déclaratif

Le paradigme déclaratif implique de décrire ce que l'on souhaite sans forcément donner tous les détails. On trouve dans cet ensemble notamment le paradigme **Fonctionnel**.

Le pseudo-code pourrait ressembler à:

```
suite = suite_de_chiffres (1,5)
Afficher suite
```

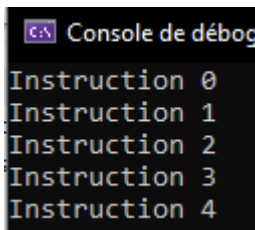
Ce code va déclarer ce qu'**est** le résultat et l'afficher en suite

Procédural

C'est ce qui est fait en 1ère année à l'ETML:

```
Console.Write("Instruction 0");
PrintRemainingInstructions();

void PrintRemainingInstructions()
{
    for(int i=1;i<5;i++)
    {
        Console.Write($"Instruction {i}");
    }
}
```



Console de débogage

Instruction 0
Instruction 1
Instruction 2
Instruction 3
Instruction 4

Chaque instruction est exécutée l'une après l'autre tout en pouvant agréger du code dans des fonctions qui sont appelées ici ou là...

Orienté objet

C'est ce qui est fait en 2ème année à l'ETML:

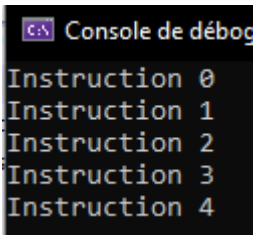
```
Console.WriteLine("Instruction 0");
for (int i = 1; i < 5; i++)
{
    Console.WriteLine(new Instruction());
}

class Instruction
{
    static int generator=1;
    public int Id { get; private set; }

    public Instruction()
    {
        Id = generator++;
    }

    public override string ToString()
    {
        return $"Instruction {Id}";
    }
}
```

```
}
```



L'idée derrière la programmation objet est de réunir les données et les fonctions au sein de classes spécifiques afin d'organiser le code de manière optimale notamment pour travailler en équipe et faciliter la modularité d'un programme...

Attention toutefois car ce paradigme peut à l'extrême devenir contre-productif car difficile à maintenir ou non indiqué pour les architectures en réseau où il est plus efficace de faire transiter les données uniquement... Cet avertissement a pour but d'éviter une polarisation de type *il faut toujours tout faire en objet*.

Fonctionnel

C'est ce qui es fait en 3ème année à l'ETML 🤖

Comme son nom l'indique, le paradigme fonctionnel fait la grande part aux fonctions.

Tout est fonction, ou presque

Ainsi, l'exemple précédent pourrait ressembler à ça:

```
Enumerable.Range(0, 5)
    .Select(i => $"Instruction {i}")
    .ToList()
    .ForEach(instruction => Console.WriteLine(instruction));
```

À noter qu'on peut mélanger les paradigmes, par exemple ci-dessous en utilisant la poo :

```
Enumerable.Range(0, 5)
    .Select(i => new Instruction())
    .ToList()
    .ForEach(instruction => Console.WriteLine(instruction));
```

Les autres

D'autres étiquettes existent pour décrire des manières de programmer, on trouve notamment:

- réactive

- logique
- concurrente
- ...