Pureté, immutabilité

Voici 2 termes en lien avec la programmation fonctionnelle qu'il s'agit de démystifier...

Pureté

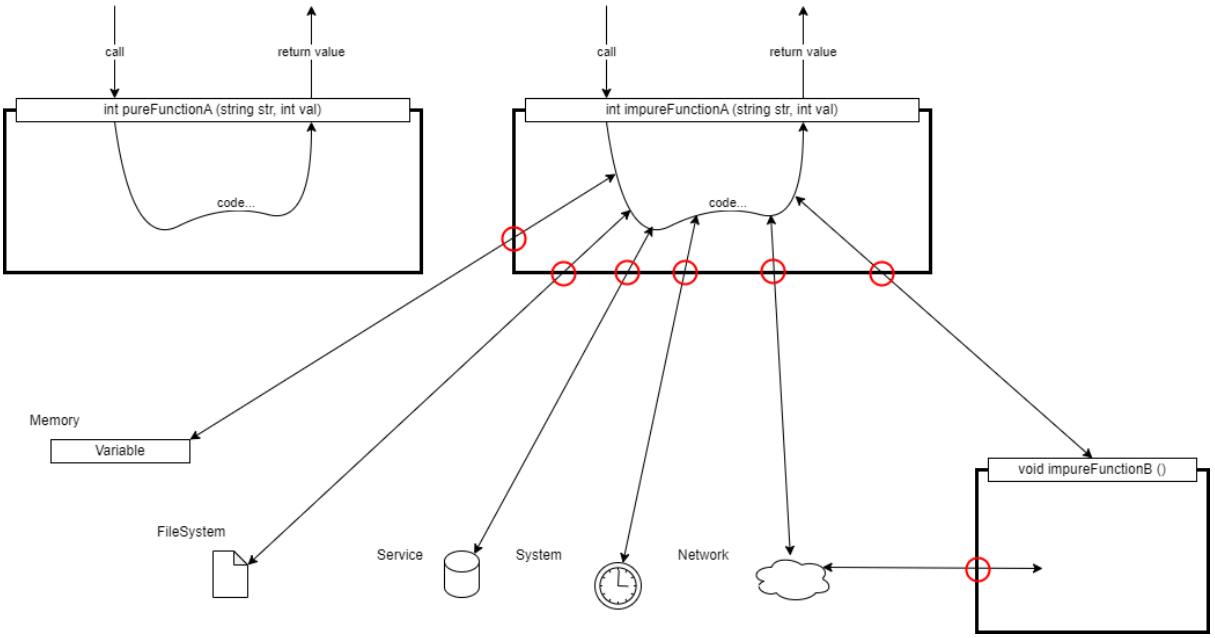
Une **fonction pure** est un concept clé en programmation fonctionnelle. Il est simple à énoncer, mais peut s'avérer difficile à appliquer.

Définition

> *Une fonction pure est une fonction qui, pour des mêmes arguments en entrée, renverra toujours le même résultat, sans provoquer d'effets secondaires.*

Ses caractéristiques sont donc:

- La prévisibilité : Une fonction pure se comporte comme une "boîte noire" mathématique. Chaque fois qu'on lui passe les mêmes valeurs, on obtient toujours le même résultat.
  Par exemple, la fonction `x => x * 2` est pure, car si on lui donne 3 en entrée, on recevra toujours 6 en sortie.
- L'absence d'effets secondaires : Une fonction pure ne modifie rien en dehors d'elle-même. Cela signifie qu'elle ne change pas de variables globales, n'écrit pas dans des fichiers ou des bases de données, ni ne modifie les objets en dehors de son propre contexte.
- L'indépendance : Le résultat que retourne une fonction pure ne dépend que des arguments qui lui sont passés. Elle ne va pas rechercher de données externes (fichiers, réseau, variable globale, ...) et elle ne fait pas appel à une fonction impure.



Les fonctions pures sont importantes car elles facilitent le débogage et les tests : il suffit de vérifier l'entrée et la sortie, sans se soucier des changements cachés. De plus, elles permettent d'écrire un code plus sûr et prévisible, car les développeurs peuvent être certains que l'exécution de ces fonctions n'affectera pas d'autres parties du programme.

Cas concret : le cache

Si une fonction est pure, cela veut dire qu'avec le même *input*, elle donne toujours le même *output*.

Ainsi, on peut facilement implémenter un **cache automatique** comme en python avec l'annotation `@cache` :

```
@cache
def factorial(n):
    return n * factorial(n-1) if n else 1
```

Pour le csharp [voici un projet intéressant]

Immutabilité

Tout aussi simple à énoncer, mais pas si simple à mettre en oeuvre, l'immutabilité se définit ainsi:

> *Une fois qu'une donnée (variable, objet, tableau, etc.) est créée, elle ne peut plus être modifiée.*

Une variable, un objet ou une structure de données immuable est "figée" dès sa création. Si on souhaite réaliser des opérations ou transformations sur cette donnée, on crée une nouvelle donnée avec la valeur souhaitée.
Par exemple, si une variable `x` vaut 5 et que l'on veut lui ajouter 3, au lieu de modifier `x`, on génère une nouvelle variable `y = x + 3`, tout en laissant `x` intacte.

L'immutabilité rend le code plus sûr et prévisible, car on sait qu'une donnée, une fois définie, ne changera pas au cours de l'exécution. Cela permet de réduire les erreurs liées à des modifications inattendues des variables ou des objets.

Attention: dans certaines situations, on peut croire à tort que l'on applique l'immutabilité:

```
static class ImmutableIntList
{
    static readonly List<int> items = new List<int>();

    public static void Add(int x) { items.Add(x); }

    public static void Dump() { items.ForEach(i => Console.WriteLine(i)); }

    public static void Reset() { items.Clear(); }
}
```

Ce code autorise de modifier le contenu de la liste:

```
items.Add(1);
items.Add(2);
items.Add(3);
```

Ce qui est immutable, c'est l'adresse de la liste en mémoire. Il n'est donc pas possible de faire:

```
items = new List<int>();
```

Parallèlisation

L'immutabilité est une aussi un moyen de *rendre obsolète* les mécanismes de `Lock` dans un cadre *multi-thread* et ainsi éviter les `deadlock` qui peuvent paralyser tout un système de par leur nature non mutable...

Cet aspect ne sera pas plus détaillé pour l'instant mais il est important de savoir qu'en tant que développeur le choix est offert de créer des espaces mémoires `mutables` et `non mutables`.

À titre d'exemple, Rust, un langage récent et très populaire (utilisé pour le kernel Linux à la place du C...) a fait le choix que, par défaut, on a une constante... à moins d'ajouter le mot clé `mut` devant pour dire que c'est une variable...