Programmation fonctionnelle

Le MapReduce

Lionel Souop

Programmation Fonctionnelle

Objectif

- Introduction aux principes de la programmation fonctionnelle
- L’exemple du MapReduce
- Programmation fonctionnelle en Python

Pré-requis

- Avoir les base en Python
- Connaissance algorithmiques de base

Programmation Fonctionnelle

Un exemple

Considérez les fonctions suivantes

```
int fact_iter(int n)
{
        int res=1,i;
        for(i=1; i<=n; i++) res = res*i;
        return res;
}
```

et

```
int fact_rec(int n)
{
        if(n<=1) return 1;
        else return n*fact_rec(n-1);
}
```

Programmation Fonctionnelle

Idées principales

- La fonction **fact_rect** est l'implémentation qui est la plus proche de la philosophie de la programmation fonctionnelle:
    - Style déclaratif vs impératif
    - Récursivité vs boucle
- Pourquoi opter pour ce style
    - Donne des programmes plus concis et clairs
    - Fournis une bonne généricité

Programmation Fonctionnelle

Un autre exemple

```
int fact (int n) { … }

int myDouble  (int n) { return 2*n; }

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

La programmation fonctionnelle demande d'avoir une fonction du style:

```
int apply_a_func (?FUNCTION? F, int n) { return f(n); }
```

Qui pourra être utiliser comme suit:

```
apply_a_func (fact, 5) // => 120

apply_a_func (myDouble, 5) // => 10

apply_a_func (square, 5) // => 25
```

Programmation Fonctionnelle

Fonctions d’ordre supérieur

- Une fonction d’ordre supérieur est une fonction qui prend comme argument au moins une autre fonction.
- intérêt:
    - Elle permet de concevoir des programmes génériques
    - Permet de programmer une seule fois l’algorithme générique, et le réutiliser pour chaque opération

Programmation Fonctionnelle

Définition

La programmation fonctionnelle est un paradigme de programmation qui s'appuie sur les fonctions d'ordre supérieur, la composition des fonctions et l'immuabilité des données, dans le but d'écrire un code déclaratif.

Les languages de programmation modernes tels que Python, Java, Scala sont adaptés pour ce style de programmation.

Programmation Fonctionnelle

MapReduce

MapReduce est un modèle de programmation fonctionnelle conçu pour le traitement distribué de grandes quantités de données sur un cluster. Il repose sur deux opérations principales :

Map → Transforme les données d'entrée en une collection de paires clé-valeur.

Reduce → Agrège les valeurs associées à chaque clé pour produire un résultat final.

Ce modèle est utilisé dans des systèmes comme Hadoop et Spark pour le traitement parallèle et scalable de données massives.

Dans ce cours, nous allons voir une version simplifiée en pure Python

Programmation Fonctionnelle

Pourquoi est-ce important dans le domaine de la data

Principe Prog fonc	Avantage en Big Data
Immuabilité	Pas de conflit d’accès concurrent
Absence d’effets de bord	Facile à paralléliser
MapReduce	Approche naturellement fonctionnelle
Expressivité	Code plus concis et lisible

Programmation Fonctionnelle

MapReduce

```
from functools import reduce

if __name__ == '__main__':
    # Appliquer une fonction sur chaque élément
    numbers = [1, 2, 3, 4, 5]
    mapped = map(lambda x: x ** 2, numbers)  # [1, 4, 9, 16, 25]
    result = reduce(lambda x, y: x + y, mapped)  # 1 + 4 + 9 + 16 + 25 = 55
    print(result)  # Output: 55
```

Programmation Fonctionnelle

MapReduce

```
def square(n):
    return n ** 2

def sum_num(x, y):
    return x + y

mapped = map(square, numbers)  # [1, 4, 9, 16, 25]
result = reduce(lambda x, y: sum_num(x,y), mapped)  # 1 + 4 + 9 + 16 + 25 =
55
print(result)  # Output: 55
```

Programmation Fonctionnelle

map()

En Python, la fonction map retourne un **iterator.**

Après avoir utiliser la fonction map, il n'est donc pas possible d'accéder directement à la collection retourner. Pour y accéder, il faut transformer l'iterator en liste, ou alors de faire une itération sur l'iterator.

```python
if __name__ == '__main__':
    # Appliquer une fonction sur chaque élément
    numbers = [1, 2, 3, 4, 5]
    mapped = map(lambda x: x ** 2, numbers)  # [1, 4, 9, 16, 25]

    # Convert to list
    result = list(mapped)
    print(result)

    #### OR ####

    # Print elements one by one
    for value in mapped:
        print(value)
```

Programmation Fonctionnelle

map()

```python
if __name__ == '__main__':
    # Appliquer une fonction sur chaque élément
    numbers = [1, 2, 3, 4, 5]
    mapped = map(lambda x: x ** 2, numbers)  # [1, 4, 9, 16, 25]

    # Convert to list
    result = list(mapped)
    print(result)

    #### OR ####

    # Print elements one by one
    for value in mapped:
        print(value)
```

map() est consommé après l'itération, donc si nous réutilisons mapped après avoir fait list(mapped), le résultat sera vide. Une bonne pratique serait de toujours transformer en list si on veut réutiliser les données.