Exercices: Programmation fonctionnelle

Lionel Vaux Auclair

Logique
HUGO@Centrale, 2021–2022

Exercice 1. Dans votre langage préféré, définissez une fonction `suite` qui prend en argument une fonction `f` sur les flottants, une valeur initiale `u0`, et un entier `n`, et qui renvoie la valeur de l'élément numéro `n` de la suite $(u_n)_{n\in\mathbf{N}}$ définie par

$$\begin{cases} u_0 = \texttt{u0} \\ u_{n+1} = \texttt{f}(u_n) \end{cases}.$$

Par exemple en Python :

```
def suite(f,u0,n):
        ...
```

ou en C :

```
int suite(float (*f)(float), float u0, int n){
        ...
}
```

et si par exemple on a une fonction `deuxfois` qui renvoie le double de son argument, alors `suite(deuxfois,1,10)` doit renvoyer 1024.

Exercice 2. Dans certains langages, on peut renvoyer des fonctions définies à l'intérieur d'autres fonctions. On peut par exemple transformer la fonction `suite` de l'exercice précédent pour qu'elle prenne seulement `f` et `u0` comme arguments et renvoie la fonction des entiers vers les flottants représentant la suite (u_n).

1. Pouvez-vous faire ça dans votre langage préféré ?
2. Pourquoi est-ce que c'est impossible (ou en tout cas nécessairement compliqué) en C ?

Exercice 3. Python et JavaScript sont deux langages impératifs, orientés objet, avec effets de bords, *etc.*, mais ils offrent aussi quelques primitives pour programmer avec un style fonctionnel. On peut par exemple écrire des fonctions anonymes avec la construction `lambda ...: ...` en Python ou `function (...) { ...}` en JS. Et pour agir sur une liste de valeurs, on peut par exemple utiliser la méthode `map` des tableaux en JavaScript, ou la primitive `map` de Python.

1. Si vous ne connaissez pas encore les fonctions anonymes ou `map`, cherchez dans votre documentation préférée.
2. Écrivez dans l'un ou l'autre langage une fonction `array_to_str_array` qui prend en argument un tableau de valeurs, et qui renvoie le tableau de leurs représentations comme des chaînes (*via* leur méthode `toString` en JS, ou la primitive `str` en Python par exemple), *en utilisant* `map`.

Par exemple en JS, on veut que

```
array_to_str_array([true,1,"toto"])
```

renvoie `[ "true", "1", "toto" ]`.

3. Écrivez dans l'un ou l'autre langage une fonction `array_adder` qui prend un argument `x`, et qui renvoie la fonction qui attend un tableau de valeurs, et qui renvoie le tableau obtenu en ajoutant `x` à toutes les valeurs.

Par exemple en Python, on veut que

```
array_adder(2)
```

soit une fonction et que

```
array_adder(2)([1,2,42])
```

renvoie `[3, 4, 44]`.

Exercice 4. Plutôt que d'utiliser des boucles `while`, la programmation fonctionnelle utilise la récursion. C'est particulièrement efficace pour parcourir des structures.

1. Écrivez une fonction récursive (sans boucles) qui recherche le plus grand élément d'un tableau.

2. Écrivez une fonction récursive (sans boucles) `passe_haut` qui prend en argument une valeur `x`, et renvoie une fonction qui attend un tableau `a`, et retourne le tableau constitué des valeurs de `a` supérieures ou égales à `x`.

La limite de cette approche est qu'on peut saturer la pile.

3. Testez les fonctions précédentes sur les tableaux de longueur 10^6 par exemple.

Les « vrais » langages de programmation fonctionnelle utilisent une optimisation dans le cas des appels récursifs terminaux : ni Python ni JS ne font ça.