Programmation fonctionnelle

Langages et programmation

Exercice 1 *QCM – Identifier le paradigme*

Pour chaque question, une seule réponse est correcte.

1. Parmi les caractéristiques suivantes, laquelle est propre au paradigme fonctionnel ?

   A. L'utilisation de boucles `for` et `while`

   B. L'encapsulation de données dans des objets

   C. La composition de fonctions et l'absence d'effets de bord

   D. La modification de variables globales

2. Quel est le résultat de `list(map(lambda x: x ** 2, [1, 2, 3]))` ?

   A. `[2, 4, 6]`

   B. `[1, 8, 27]`

   C. `[1, 2, 3]`

   D. `[1, 4, 9]`

3. Qu'est-ce qu'une fonction d'ordre supérieur ?

   A. Une fonction qui prend une autre fonction en paramètre ou qui renvoie une fonction

   B. Une fonction qui s'appelle elle-même

   C. Une fonction définie à l'intérieur d'une autre fonction

   D. Une fonction qui modifie une variable globale

4. Quelle expression Python est écrite en style fonctionnel ?

   A. `result = 0 ; for x in L: result += x`

   B. `while i < len(L): i += 1`

   C. `sum(filter(lambda x: x > 0, L))`

   D. `obj.calculer(L)`

Exercice 2 *Reconnaître le paradigme*

Pour chaque extrait de code, indiquer s'il est écrit en style **impératif**, **fonctionnel** ou **objet**. Justifier.

a)

```
def somme(L):
    total = 0
    for x in L:
```

```
        total += x
    return total
```

b)

```
def somme(L):
    return 0 if L == [] else L[0] + somme(L[1:])
```

c)

```
class Calcul:
    def __init__(self, L):
        self.L = L
    def somme(self):
        return sum(self.L)
```

d)

```
from functools import reduce
somme = lambda L: reduce(lambda acc, x: acc + x, L, 0)
```

Exercice 3 *map filter et lambda (exercice guidé)*

a) Complétez les expressions suivantes (notées `...`) :

```
L = [3, 7, 2, 9, 4]

# Doubler chaque élément
list(map(lambda x: ..., L))          # → [6, 14, 4, 18, 8]

# Garder seulement les éléments impairs
list(filter(lambda x: ..., L))       # → [3, 7, 9]

# Convertir en chaînes
list(map(..., L))                     # → ['3', '7', '2', '9', '4']
```

b) Écrivez les expressions pour obtenir, à partir de `L = [3, 7, 2, 9, 4]` :

1. la liste des carrés : `[9, 49, 4, 81, 16]` ;
2. la liste des éléments pairs : `[2, 4]` ;
3. la liste des éléments supérieurs à 5 : `[7, 9]`.

Donnez chaque réponse avec `map`/`filter` + `lambda`, puis avec une compréhension de liste.

c) À partir de `mots = ['Python', 'est', 'un', 'langage']`, obtenir la liste des longueurs : `[6, 3, 2, 7]`.

d) À partir de `L = [-3, 5, -1, 8, -2, 7]`, obtenir la liste des valeurs absolues des éléments négatifs : `[3, 1, 2]` (combinaison de `filter` et `map`).

Exercice 4 *sorted avec key et lambda*

a) Trier la liste `mots = ['Python', 'est', 'un', 'langage', 'multiparadigme']` par longueur croissante.

b) Trier la liste `eleves = [('Alice', 15), ('Bob', 12), ('Clara', 18), ('David', 14)]` par note décroissante.

c) Trier la liste `points = [(1, 5), (3, 2), (0, 0), (4, 1)]` par distance croissante à l'origine.

Exercice 5 *Réécriture impératif vers fonctionnel*

Réécrire chaque fonction en style fonctionnel (pas de variable modifiable, pas de boucle, récursivité uniquement).

a) Longueur d'une liste :

```
def longueur(L):
    compteur = 0
    for _ in L:
        compteur += 1
    return compteur
```

b) Appartenance d'un élément à une liste :

```
def appartient(x, L):
    for elem in L:
        if elem == x:
            return True
    return False
```

c) Nombre d'occurrences :

```
def nb_occurrences(x, L):
    compteur = 0
    for elem in L:
        if elem == x:
            compteur += 1
    return compteur
```

Exercice 6 *Fonctions pures et effets de bord*

Pour chaque fonction, indiquer si elle est **pure** ou **impure**. Justifier.

a)

```
def double(x):
    return 2 * x
```

b)

```
compteur = 0
def incrementer():
    global compteur
    compteur += 1
```

```
    return compteur
```

c)

```
def ajouter(x, L):
    L.append(x)
    return L
```

d)

```
def ajouter_pur(x, L):
    return L + [x]
```

Exercice 7 *La fonction `forall` (exercice guidé)*

On souhaite écrire une fonction `forall(pred, L)` qui renvoie `True` si tous les éléments de `L` vérifient le prédicat `pred`.

Données de test :

```
pair = lambda x: x % 2 == 0
ex1 = [4, 4, 2, 0, 1]   # attendu : False
ex2 = []                  # attendu : True
ex3 = [4, 4, 2, 0, 8]   # attendu : True
```

a) Écrivez `forall` avec une boucle `for`. C'est la version la plus naturelle : dès qu'un élément ne vérifie pas le prédicat, on renvoie `False`.

b) Écrivez `forall` en version récursive pure (sans boucle, sans variable). Quel est le cas de base ? Comment se réduit le problème ?

c) Écrivez `forall` avec `map` et `all`. Quel est le rôle de chacune de ces fonctions ?

d) Écrivez `forall` avec une compréhension de liste et `all`.

e) Écrivez `forall` avec `filter` et `len`.

Exercice 8 *Synthèse – Trois paradigmes sur un même problème*

On dispose d'une liste d'enregistrements représentant des élèves :

```
eleves = [
    {"nom": "Alice", "notes": [15, 12, 18]},
    {"nom": "Bob", "notes": [8, 14, 9]},
    {"nom": "Clara", "notes": [16, 17, 19]},
    {"nom": "David", "notes": [11, 7, 13]},
]
```

On souhaite obtenir la liste des noms des élèves dont la moyenne est supérieure ou égale à 14.

Résultat attendu : `['Alice', 'Clara']`

a) Écrire une solution en style **impératif** (boucle, variable accumulatrice).

b) Écrire une solution en style **fonctionnel** (avec `filter`, `map`, `lambda`), puis en compréhension de liste.

c) Écrire une solution en style **objet** (avec une classe `Eleve` disposant des méthodes `moyenne` et `est_bon`).

d) Comparer les trois approches : laquelle est la plus lisible ? La plus courte ? La plus facilement testable ?