

Programmation fonctionnelle

Fiche de synthèse

① LE PARADIGME FONCTIONNEL

Le premier langage fonctionnel, Lisp, date de 1958; OCaml et Haskell en sont des descendants. Python n'est pas un langage fonctionnel pur, mais on peut y programmer dans un **style fonctionnel**.

Définition 1 (Style fonctionnel) Un programme en **style fonctionnel** calcule son résultat avec des fonctions, la composition de fonctions, des expressions conditionnelles et la récursivité, sans variable dont la valeur change, sans boucle et sans instruction exécutée pour son effet.

```
# style impératif
def nombre_de_chiffres(n):
    chiffres = 1
    while n >= 10:
        n = n // 10
        chiffres = chiffres + 1
    return chiffres

# style fonctionnel
def nombre_de_chiffres(n):
    return (1 if n < 10 else
            1 + nombre_de_chiffres(n // 10))
```

- L'**expression conditionnelle** `v1 if condition else v2` vaut `v1` ou `v2`, alors qu'une instruction `if` ne vaut rien.
- La **composition** utilise le résultat d'une fonction comme argument d'une autre : `somme(reste(liste))`.

Impératif	Fonctionnel
enchaîne des instructions	compose des fonctions
exécute des instructions	calcule des expressions
répète avec une boucle	rappelle la fonction

② FONCTIONS PURES ET EFFETS DE BORD

Définition 2 (Fonction pure) Une fonction est **pure** quand son résultat ne dépend que de ses arguments et qu'elle ne modifie rien en dehors d'elle-même. Une modification extérieure (variable globale, liste reçue en argument, fichier, écran) s'appelle un **effet de bord**.

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

```
def ajouter_pur(x, L):     # pure : nouvelle liste
    return L + [x]
```

En programmation fonctionnelle, on construit une nouvelle structure plutôt que de modifier une structure existante.

③ FONCTIONS D'ORDRE SUPÉRIEUR

Définition 3 (Fonction d'ordre supérieur) Une fonction est **d'ordre supérieur** quand elle reçoit une fonction en argument ou renvoie une fonction.

Le mot-clé `lambda` crée une fonction sans nom : `double = lambda x: 2 * x`, puis `double(16)` vaut 32.

```
villes = ['Bordeaux', 'Paris', 'Lyon']
sorted(villes, key=len)
# ['Lyon', 'Paris', 'Bordeaux']
eleves = [("Inès", 14), ("Hugo", 9)]
sorted(eleves, key=lambda e: e[1])
# [('Hugo', 9), ('Inès', 14)]
```

`sorted` renvoie une nouvelle liste triée; avec `key`, elle range les éléments selon leur image par la fonction donnée.

```
list(map(lambda x: x ** 3, [3, 8, 2]))
# [27, 512, 8]
list(filter(lambda x: x > 4, [3, 7, 2, 9]))
# [7, 9]
```

- `map(f, liste)` applique `f` à chaque élément.
- `filter(predicat, liste)` garde les éléments pour lesquels le **prédicat**, fonction qui renvoie un booléen, vaut `True`.
- Les deux renvoient un **itérateur**, qui calcule les valeurs à la demande; `list` le transforme en liste.
- Une liste en compréhension donne les mêmes résultats : `[x ** 3 for x in L]` et `[x for x in L if x > 4]`.

Remarque 1. Hors programme. `reduce(f, liste, depart)`, du module `functools`, combine tous les éléments en une seule valeur : `reduce(lambda acc, x: acc + x, [3, 7, 2], 0)` calcule `0 + 3`, puis `3 + 7`, puis `10 + 2` et vaut 12.

④ TROIS PARADIGMES

Pour calculer la moyenne des notes supérieures ou égales à 10, le style **impératif** parcourt la liste avec une boucle et met à jour une somme et un compteur; le style **objet** écrit une

classe `ReleveDeNotes` munie d'une méthode `moyenne_au_dessus` ; le style **fonctionnel** compose des fonctions :

```
def moyenne_au_dessus(notes):
    garde = list(filter(lambda n: n >= 10,
                        notes))
    return (sum(garde) / len(garde) if garde
            else 0)
```

	Impératif	Fonctionnel	Objet
Brique	instruction, variable	fonction, expression	objet, méthode
Contrôle	boucles, conditions	récurtivité, composition	messages entre objets
État modifié	oui	non	oui, les attributs
Langage	C	Haskell	Smalltalk

Python est **multiparadigme** : on peut y combiner les trois styles.

À retenir.

- Le style fonctionnel compose des fonctions et répète par récursivité, sans modifier de variable.
- Une fonction pure ne dépend que de ses arguments et n'a aucun effet de bord.
- Une fonction d'ordre supérieur reçoit ou renvoie une fonction : `map`, `filter`, `sorted` avec `key`.
- `lambda` crée une fonction sans nom, souvent passée en argument.