NSI · TERMINALE / COURS

Les paradigmes de programmation

Astrolab Cours · astrolabcours.fr

01 Le problème

On peut calculer la somme des carrés des nombres pairs d'une liste de plusieurs façons : avec une boucle et une variable qu'on modifie, avec des fonctions qu'on enchaîne, ou avec des objets. Ce ne sont pas des détails de style : ce sont des manières différentes de penser un programme.

01 L'idée : trois façons de donner une recette

- La recette **impérative** : « prends un bol, verse la farine, ajoute les œufs, mélange ». Une suite d'ordres qui modifient l'état de la cuisine.
- La recette **fonctionnelle** : « la pâte, c'est mélanger(farine, œufs) ». On décrit ce qu'on obtient en combinant des transformations.
- La recette **objet** : on a un objet Four, avec ses réglages et sa méthode `cuire`. On fait interagir des objets.

01 Le paradigme impératif

Le programme est une **suite d'instructions** qui modifient l'**état** (les variables).

```
s = 0
for x in liste:
    if x % 2 == 0:
        s = s + x ** 2
```

01 Le paradigme fonctionnel

Le programme est une **composition de fonctions**. Les fonctions sont **pures** : même entrée, même sortie, et **aucun effet de bord** (rien n'est modifié hors de la fonction).

```
pairs = filter(lambda x: x % 2 == 0, liste)
carres = map(lambda x: x ** 2, pairs)
s = sum(carres)
```

Les fonctions sont des valeurs comme les autres : on peut les **passer en paramètre** (`map` reçoit une fonction) ou les renvoyer.

Une fonction pure est facile à tester et à réutiliser : son résultat ne dépend que de ses paramètres.

© Astrolab Cours · astrolabcours.fr/nsi/terminale/paradigmes-de-programmation/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

Le paradigme objet

Le programme est un ensemble d'**objets** qui regroupent données et méthodes, et qui s'envoient des messages (voir le chapitre sur la programmation objet).

Choisir

Paradigme	Point fort
impératif	proche du fonctionnement de la machine, intuitif
fonctionnel	pas d'effets de bord, facile à tester, se parallélise bien
objet	modéliser des entités (personnages, comptes, fenêtres)

Python permet les trois. Haskell est surtout fonctionnel ; C est impératif ; Java est très orienté objet.

Exemple guidé

Calcule la somme des carrés des nombres pairs de `L = [1, 2, 3, 4]` dans deux paradigmes. Le résultat attendu est $2^2 + 4^2 = 20$.

```
# impératif : une variable qu'on modifie
total = 0
for x in L:
    if x % 2 == 0:
        total = total + x * x

# fonctionnel : des fonctions qu'on enchaîne
total = sum(map(lambda x: x * x,
                filter(lambda x: x % 2 == 0, L)))
```

1. **Impératif** : on décrit les étapes, et l'état (`total`) change à chaque tour.
2. **Fonctionnel** : on décrit une transformation (filtrer, élever au carré, sommer). Aucune variable n'est modifiée, et `L` reste intacte.
3. En Python, la forme la plus lisible mélange les deux : `sum(x * x for x in L if x % 2 == 0)`.

Retour au problème

Le même calcul s'écrit comme une suite d'ordres ou comme une composition de fonctions. L'objet, lui, servirait si ces nombres appartenaient à des entités avec leurs propres méthodes, comme des comptes ou des personnages.

Les pièges

- Une fonction qui affiche, modifie une variable globale ou une liste reçue a un effet de bord.
- `map` et `filter` renvoient des objets à parcourir, pas des listes : `list(map(...))` pour les voir.
- Les paradigmes ne s'excluent pas : un même programme peut les mélanger.

© Astrolab Cours · astrolabcours.fr/nsi/terminale/paradigmes-de-programmation/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits