NSI · TERMINALE / COURS

La récursivité

Astrolab Cours · astrolabcours.fr

01 Le problème

Pour calculer $5! = 5 \times 4 \times 3 \times 2 \times 1$, on peut écrire une boucle. Mais on peut aussi remarquer que $5! = 5 \times 4!$. Le problème contient une version plus petite de lui-même. Comment l'exploiter dans un programme ?

01 L'idée : les poupées russes

Pour savoir combien il y a de poupées, tu ouvres la première : il y en a une de plus que dans celle qu'elle contient. Tu recommences avec la suivante, jusqu'à la plus petite, qui ne s'ouvre pas. C'est elle qui donne la première réponse, et toutes les autres se calculent en remontant.

01 Une fonction qui s'appelle elle-même

Une fonction est **récursive** quand elle s'appelle elle-même. Elle contient toujours :

- un **cas de base**, traité directement, sans appel récursif (la plus petite poupée) ;
- un **appel récursif** sur un problème **plus petit**, qui se rapproche du cas de base.

```
def factorielle(n):
    if n == 0:          # cas de base
        return 1
    return n * factorielle(n - 1)   # appel récursif
```

La pile d'appels

Quand `factorielle(3)` s'exécute, elle attend le résultat de `factorielle(2)`, qui attend celui de `factorielle(1)`, puis de `factorielle(0)`. Ces appels en attente sont empilés dans la **pile d'appels**. Le cas de base répond 1, puis les calculs se font en remontant : $1 \times 1 = 1$, $2 \times 1 = 2$, $3 \times 2 = 6$.

Chaque appel en attente occupe de la mémoire. Python limite la pile à environ 1 000 appels : au-delà, il lève une `RecursionError`.

Un exemple plus malin : la puissance rapide

$x^n = (x^{n/2})^2$ si n est pair, et $x^n = x \times x^{n-1}$ sinon.

```
def puissance(x, n):
    if n == 0:
```

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

```
        return 1
    if n % 2 == 0:
        p = puissance(x, n // 2)
        return p * p
    return x * puissance(x, n - 1)
```

Pour $n = 1\,000$, on fait une vingtaine d'appels au lieu de mille multiplications.

Prouver que ça s'arrête

Pour montrer qu'une fonction récursive termine, on trouve une quantité entière positive qui **diminue strictement** à chaque appel (ici n) : elle finit forcément par atteindre le cas de base.

Les pièges

- Oublier le cas de base, ou un appel qui ne s'en rapproche pas (`f(n + 1)`) : la récursion ne s'arrête pas.
- Oublier le `return` devant l'appel récursif : la fonction renvoie `None`.
- Récursif ne veut pas dire plus rapide : la version naïve de Fibonacci recalcule mille fois les mêmes valeurs.

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