
RÉCURSIVITÉ

Écrire un programme récursif

Exercice 0 Donner une définition récursive qui correspond au calcul de la fonction factorielle $n!$ définie par $n! = 1 \times 2 \times \dots \times n$ et $0! = 1$, puis le code d'une fonction `fact(n)` qui implemente cette définition.

Exercice 1

- Écrire (sans boucles) une fonction récursive `nb_diviseurs(j, n)` qui renvoie le nombre d'entiers entre 1 et j qui divise n .
- Écrire (sans boucles) une fonction `est_parfait` qui renvoie `True` si et seulement si la somme des diviseurs de n est égale à $2n$: les entiers parfaits entre 1 et 30 sont 6 et 28.

Exercice 2 Le fichier `exo_tri_insertion.py` contient l'implémentation du tri par insertion vue en première. Implémenter une version récursive de cet algorithme en commençant par identifier les paramètres de l'appel récursif et le cas de base.

Exercice 3 Soit $u(n)$ la fonction définie sur \mathbb{N} par

$$u(n+1) = \begin{cases} u(n)/2 & \text{si } u(n) \text{ est pair,} \\ 3u(n) + 1 & \text{sinon} \end{cases}$$

avec u_0 un entier quelconque plus grand que 1.

Écrire une fonction récursive `syracuse(u_0)` qui affiche les valeurs successives de la fonction $u(n)$ tant que $u(n)$ est plus grand que 1.

La conjecture de Syracuse affirme que, quelle que soit la valeur de u_0 , il existe une valeur de n tel que $u(n) = 1$. Cette conjecture défie toujours les mathématiciens.

Exercice 4 **PLUS ULTRA** Un mot codé en Morse est constitué d'une suite de points et de tirets. Un tiret dure 3 unités, un point 1 unité et l'espace entre les signaux (points et tirets successifs) dure 1 unité. On considérera qu'il existe un unique mot de longueur 0.

- On suppose que n'importe quelle suite de points et de tirets est un message valide (contrairement au véritable Morse). Donner une formule récursive définissant le nombre de messages $f(n)$ de durée n (sans compter les espaces) que l'on peut former sans compter les espaces.
- Même question, mais en comptant les espaces dans la durée.

Analyser le fonctionnement d'un programme récursif

Exercice 5 Que calcule chaque fonction suivante et à quel coût ?

```

1 def f1(n):
2     """ entrée: un entier n > 0 """
3     if n == 1:
4         | return 0
5     else:
6         | return f1(n//2)+1
7 def f2(n):
8     """ entrée: un entier n > 0 """
9     if n == 0:
10        | return 0
11    else:
12        | return f2(n-1)+n

```

Exercice 6 Les deux fonctions `f1_rec` et `f2_rec` ci-dessous font les mêmes opérations, mais l'instruction `print()` est placée différemment. Sans exécuter le code sur une machine :

1. Qu'affiche `f(4)` ?
2. Que se passe-t-il pour `f(1)` ?

```
1 def f1_rec(i):
2     | if i == n:
3     | | print('cas de base:')
4     | else:
5     | | print(i)
6     | | f1_rec(i+1)
7 def f2_rec(i):
8     | if i == n:
9     | | print('cas de base:')
10    | else:
11    | | f2_rec(i+1)
12    | | print(i)
13 def f(n):
14    | f1_rec(2)
15    | f2_rec(2)
```