Algorithmique - complément : la récursivité

On peut écrire des fonctions capables de s'appeler elles-mêmes.
Elles traduisent en général l'idée mathématique de récurrence.

- Initialisation $\longleftrightarrow$ Condition d'arrêt (très important pour éviter les boucles infinies)
- Hérédité $\longleftrightarrow$ Appel de la fonction elle-même à un rang inférieur.

Exemple

Puissance naïve : $x^0 = 1$ et si $n \geqslant 1$, $x^n = x \times x^{n-1}$.

$puissrec(x, n)$
- **Si** $n = 0$ **Alors**
  - **Retourner** 1
- **Sinon**
  - **Retourner** $x * puissrec(x, n-1)$
- **FinSi**

En Python :

```
def puissrec(x,n):
    if n==0: return 1
    else: return x*puissrec(x,n-1)
```

- **Principe** :

  L'interpréteur empile les résultats intermédiaires pour pouvoir faire les calculs. (Attention à la complexité spatiale, donc...)

	—	
puissrec(x,0)	`1`	
↑	—	↓
puissrec(x,1)	`x*`	
↑	—	↓
⋮	⋮	⋮
↑	—	↓
puissrec(x,n-1)	`x*`	
↑	—	↓
puissrec(x,n)	`x*`	
	—	

  Pour la terminaison, ici, notre n décroit strictement et finit par atteindre la valeur 0 (condition d'arrêt).

- **Avantages** : Facile à écrire.
- **Inconvénient** : Demande de la place en mémoire et limité à 1000 appels récursifs en Python, modifiable avec

  `sys.setrecursionlimit(nb).`

  Peut coûter très cher (en temps et espace) si mal employé !
- **Correction** : par récurrence !
- **Complexité** : Les calculs de complexité font apparaître des suites récurrentes. $T(n) = f(T(n-1))$ ou autre... Pour l'exemple précédent, c'est facile : $T(n+1) = T(n) + 1$...

Exemples

E 1 – Syracuse

E 2 – Factorielle

E 3 – Approximation de $\sqrt{2}$ par la méthode de Héron : $u_0 = 1$ et $u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right)$. On peut montrer que $u_n \to \sqrt{2}$ très rapidement.

E 4 – Exponentiation rapide

E 5 – Fibonacci

E 6 – Recherche dichotomique

E 7 – Algorithme d'Euclide

E 8 – Palindrome