La récursivité

I- Deux types de programmation

Instruction itérative	Instruction récursive
Instructions utilisant des boucles while et des boucles for.	Une fonction f est récursive si elle intervient dans sa définition.

Voici la structure canonique d'une boucle while :
`solution = >initialisation`
`while (pb pas fini)`
`    on effectue un calcul`
`    solution = nouvelle solution (liée au calcul fait)`
`    si pas fini : continue`
	Structure canonique d'un programme récursif
	`def f(pb) :`
	`    si le pb est trivial`
	`    alors solution évidente`
	`    sinon on décompose en sous pb :pb1,pb2,pb3,…`
	`    puis appliquer la fonction f(pb1),f(pb2),f(pb3),…`

Pour que le programme se termine : Il faut repérer un variant de boucle dont la valeur converge après chaque itération vers la condition d'arrêt.	Pour que le programme se termine : La variable passée en argument (typiquement un entier) décroit à chaque appel récursif et qui prend fatalement la valeur aboutissant au problème trivial (cas d'arrêt)

II- Pile, récursivité et complexité spatiale

Prenons l'exemple de la multiplication :$x \times y = x + x \times (y-1)$ avec $y > 0$

```python
def mult(x,y):
    if y == 0:
        return 0
    else :
        return x+mult(x,y-1)
```

Chaque appel de la fonction récursive qui n'est pas un cas d'arrêt est stocké en mémoire. Dès que l'on tombe sur un cas d'arrêt, le dernier calcul mis en mémoire est obtenu et le résultat est utilisé pour le calcul suivant et ainsi de suite.

Appel de `mult(6,3)` →	`mult(6,3)=6+ mult(6,2)`	
Appel de `mult(6,2)` →	`mult(6,2)=6+ mult(6,1)` `mult(6,3)=6+ mult(6,2)`	

Appel de `mult(6,1)` →	`mult(6,1)=6+ mult(6,0)`
	`mult(6,2)=6+ mult(6,1)`
	`mult(6,3)=6+ mult(6,2)`	

Appel de `mult(6,0)` →	`mult(6,1)=6+ mult(6,0)`
	`mult(6,2)=6+ mult(6,1)`
	`mult(6,3)=6+ mult(6,2)`
		Cas d'arrêt qui `mult(6,0)=0`

Calcul de `mult(6,1)` = 6	`mult(6,2)=6+ mult(6,1)` `mult(6,3)=6+ mult(6,2)`	
Calcul de `mult(6,2)` = 12	`mult(6,3)=6+ mult(6,2)`	
Calcul de `mult(6,3)` = 18		

Terminaison du programme	Les appels successifs `mult(x,y-1)` aboutissent bien au cas d'arrêt si $y > 0$.

Phénomène de dépassement	- A chaque appel récursif, de la mémoire doit être allouée pour stocker le calcul à faire
	Il faut éviter de dépasser la taille maximale accordée en mémoire, typiquement 2000 :
	`mult(5,2000)#10 000`
	`mult(5,3000)#maximum recursion depth exceeded in comparison`
	`mult(3000,5)#15000`

Bilan sur la récursivité :

Avantage : Programmation courte et élégante si elle se prête bien à la récurrence	Inconvénient : Programmation souvent non optimisé en mémoire

Commenté [AM1]: https://www.youtube.com/watch?v=_xk9Sd10VBw

Commenté [AM2]: Quel type de programmation un informaticien doit-il adopter ?
Cela dépend des situations :
La récursivité offre une lecture plus aisée, mais sa complexité spatiale est en général plus gourmande mais la décomposition en sous-problèmes équivalent favorise la résolution en parallèle sur plusieurs cœurs
Les programmes itératifs sont plus ardus à relire mais ont une meilleure complexité spatiale.
Le but de la récursivité est de programmer de façon intuitive et courte. Cependant, si le programme est court, son temps d'exécution ne l'est pas toujours.

Commenté [AM3]: Avec les boucles for, on a une situation bornée ce qui correspond tout à fait à la manipulation de tableaux numpy dont on connaît les dimensions à la création.
A l'inverse, les listes peuvent être vues comme des structures récursives $liste[:] = liste[0] + liste[1:]$: les listes sont donc tout à fait indiquées dans l'emploi des fonctions récursives dont le nombre d'itérations dépend du cas d'étude.

Commenté [AM4]: On distingue donc l'appel principal (1e appel) des appels récursifs suivant

Commenté [AM5]: On peut prendre l'exemple des tours de Hanoï :

On s'aperçoit que résoudre ce problème pour n pièces se ramène à savoir résoudre la situation pour $n-1$ pièces. De même savoir résoudre ce problème pour $n-1$ pièces revient à savoir résoudre le pb pour $n-2$ pièces….
Finalement pour construire la solution pour n disques, il suffit de savoir résoudre la solution triviale pour un disque.

Commenté [AM6]: On parle de terminaison

Commenté [AM7]: Ce variant de boucle est souvent lié à la taille du tableau manipulé

Commenté [AM8]: Plus exactement dans une pile (structure que nous étudierons plus tard)