Chapitre D.1 : La récursivité

I. Introduction

Un algorithme récursif _____

_____

Une fonction récursive _____

_____

L'approche récursive d'un problème est un concept de base en algorithmie

II. Caractèristiques d'une fonction récursive

Pour s'assurer qu'une fonction récursive soit correctement écrit, il faut vérifier qu'elle s'arrête dans tous les cas. Pour cela, il faut respecter deux règles :

- Dans une fonction récursive, il faut au moins un cas dans lequel _____. Ce cas est appelé _____.
- Chaque appel récursif doit se faire avec des données qui permettent de _____ _____. C'est à dire du _____

Exemple: La fonction factorielle.
La fonction factorielle est une fonction mathématique. Le fonction factorielle de n se note $n!$.
Pour tout entier $n > 0$ on a :
$n! = 1 \times 2 \times 3 \times ... \times n - 1 \times n$
La fonction `fact(n)` suivante renvoie la valeur de $n!$.

```
def fact(n):
  assert n > 0
  if n <= 1:
      return 1
  else:
      return fact(n-1)
```

- **Surligner le(s) cas de base**
- **Entourer le(s) appel(s) récursif(s)**

Pour vérifier qu'une fonction récursive s'arrête, il faut comme dans le cas d'une boucle while, trouver un variant de boucle permettant de démontrer que l'on s'approche d'un cas de base.
Par exemple, dans le cas de notre fonction `fact(n)`, _____

_____

_____

III. Pile d'exècution

Dans un programme, lors de l'appel d'une fonction, l'environnement de cette fonction est mis dans un espace mémoire particulier qui s'appelle la pile d'exécution. **Cet espace est limité, ce qui signifie que le nombre de fonction qui peuvent s'exécuter en même temps possède une limite à ne pas dépasser.** De plus cet espace réservé à l'exécution des fonctions fonctionne comme une pile : une fonction appelante ne pourra pas libérer sa place dans la pile tant que les fonctions appelé n'ont pas été fermée.

Exemple de l'évolution de la pile d'exècution avec la fonction `fact(n)`

On appelle `fact(4)`

appel	renvoie
fact(4)	

Dans l'appel de la fonction, on appelle `fact(3)`

appel	renvoie
fact(3)	
fact(4)	

Dans l'appel de la fonction, on appelle `fact(2)`

appel	renvoie
fact(2)	
fact(3)	
fact(4)	

Dans l'appel de la fonction, on appelle `fact(1)`. Cet appel renvoie la valeur 1.

appel	renvoie
fact(1)	1
fact(2)	
fact(3)	
fact(4)	

L'appel de `fact(1)` est terminé, on reprend l'exècution de `fact(2)`. Cet appel renvoie la valeur 2.

appel	renvoie
fact(2)	2
fact(3)	
fact(4)	

L'appel de `fact(2)` est terminé, on reprend l'exècution de `fact(3)`. Cet appel renvoie la valeur 6.

appel	renvoie
fact(3)	6
fact(4)	

L'appel de `fact(3)` est terminé, on reprend l'exècution de `fact(4)`. Cet appel renvoie la valeur 24.

appel	renvoie
fact(4)	24

IV. Limites de la récursivité

Les algorithmes écrit de façon récursif sont plus lisibles et évitent l'utilisation de nombreuses boucles. Cependant, la récursivité est gourmande en ressources mémoires et risque de provoquer un débordement

lorsque la pile d'exécution est pleine.

Tous les algorithmes récursifs peuvent être écrit de façon itérative.

Exemple : La fonction factorielle écrit de façon itérative

```
def fact_iteratif(n):
    res = 1
    for i in range(1,n+1):
        res = res * i
        return res
```