? À rendre LUNDI 5 janvier 2026

Devoir Maison n°5 – *Corrigé*

Partie I Travail à faire

Partie II Et le range dans tout ça ?

R1. Je dois parcourir une liste `L` entièrement.

(a) Quel est le premier rang ? Quelle est la valeur minimale de `i` pour `L[i]` ?

Solution: Premier rang : 0 ; Valeur minimale de i : 0

(b) Quel est le dernier rang ? Quelle est la valeur maximale de `i` pour `L[i]` ?

Solution: Dernier rang : len(L)-1 ; Valeur maximale de i : len(L)-1

(c) Dans la boucle où j'accède à `L[i]` uniquement, qu'indiquer dans le `range` ?

Solution: `range(0,len(L))` ou `range(len(L))`

R2. Je dois parcourir une liste `L` entièrement, dans la boucle je dois accéder à `L[i]` et `L[i-1]`.

(a) Pour `L[i]` : quelle est la valeur minimale que peut prendre `i` ? quelle est la valeur maximale que peut prendre `i` ?

Solution: $0 \leq i \leq len(L) - 1$

(b) Pour `L[i-1]` :
Quelle est la valeur minimale que peut prendre `i-1` ? et donc `i` ?
Quelle est la valeur maximale que peut prendre `i-1` ? et donc `i` ?

Solution: $0 \leq i - 1 \leq len(L) - 1$, donc $1 \leq i \leq len(L)$

(c) Par conséquent, compléter la boucle `for` suivante permettant de parcourir toute la liste sans en sortir : `for i in range(...,...):`

Solution: Dans la boucle, on accède à la fois à `L[i]` et `L[i-1]`, donc $1 \leq i \leq len(L) - 1$, donc $1 \leq i < len(L)$: `for i in range(1,len(L)):`

R3. Je dois parcourir une liste `L` entièrement, dans la boucle je dois accéder à `L[i]` et `L[i+1]`.

(a) Pour `L[i]` : quelle est la valeur minimale que peut prendre `i` ? quelle est la valeur maximale que peut prendre `i` ?

Solution: $0 \leq i \leq len(L) - 1$

(b) Pour `L[i+1]` :
Quelle est la valeur minimale que peut prendre `i+1` ? et donc `i` ?
Quelle est la valeur maximale que peut prendre `i+1` ? et donc `i` ?

Solution:
$0 \leq i + 1 \leq len(L) - 1$, donc $-1 \leq i \leq len(L) - 2$

(c) Par conséquent, compléter la boucle `for` suivante permettant de parcourir toute la liste sans en sortir : `for i in range(...,...):`

> **Solution:** Dans la boucle, on accède à la fois à `L[i]` et `L[i+1]`, donc, $0 \leq i \leq len(L) - 2$, donc $0 \leq i < len(L) - 1$, donc `for i in range(0,len(L)-1):` (pour aller jusqu'à `len(L)-2` inclus.

Partie III Révisions sur la récursivité

À retenir

Une fonction récursive est une fonction qui s'appelle elle-même.

```
def f_rec(a,b,c):
    if a==25 : # condition d'arrêt qui porte sur a ou b ou c
        return 0 # valeur pour a=25
    else :
        return f(a+1,b,c) # appel récursif de f_rec sur un rang
   précédent de a ou b ou c
```

R1. Écrire la fonction récursive `factorielle(n:int)->int` qui renvoie la valeur de $n!$, en exploitant le fait que $n! = n \times (n-1)!$, et $0! = 1$.

Solution:

```
def factorielle(n):
    if n==0 :  # condition d'arrêt
        return 1
    else :
        return n*factorielle(n-1) # appel récursif
```

Comment ça fonctionne ? Les instructions en attente sont empilées, avant d'être dépilées (la structure informatique derrière est une pile).
> `factorielle(3)` demande `3*factorielle(2)`
– – > `factorielle(2)` demande `2*factorielle(1)`
– – – > `factorielle(1)` demande `1*factorielle(0)`
– – – > calcul de 1!
– – > calcul de 2!
> calcul de 3!

R2. Écrire la fonction récursive `fibo(n:int)->int` qui renvoie la valeur de F_n de la suite de Fibonacci définie par $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$.

Solution:

```
def fibo(n):
    if n==0 : #  condition d'arrêt
        return 0
    elif n==1 : # condition d'arrêt
        return  1
    else :
        return fibo(n-1)+fibo(n-2)  # appel récursif
```

R3. Une suite (c_n) plus subtile : $c_0 = 2$ et $c_{n+1} = \begin{cases} \frac{c_n}{2} & \text{si} \quad c_n \text{ est pair} \\ 3c_n + 1 & \text{si} \quad c_n \text{ est impair} \end{cases}$

Que l'on peut réécrire, pour $n \geq 1$: $c_n = \begin{cases} \dfrac{c_{n-1}}{2} & \text{si} \quad c_{n-1} \text{ est pair} \\ 3c_{n-1}+1 & \text{si} \quad c_{n-1} \text{ est impair} \end{cases}$

Rappel, `n%2` renvoie le reste de la division euclidienne par 2.

Solution:

```
def suite(n):
    if n==0 : #  condition d'arrêt
        return 2
    else :
        prec = suite(n-1) # valeur de c_(n-1) récupéré via appel récursif
        if prec%2==0 :# si pair
            return prec//2
        else :
            return 3*prec+1
```

R4. Un tri récursif : le tri fusion. On suppose avoir une fonction `fusion(L1:list,L2:list)->list` qui à partir des deux listes `L1` et `L2` triées dans le même ordre, renvoi une liste triée dans le même ordre.

Écrire une fonction récursive `tri_fusion(L:list)->list` qui trie par ordre croissant la liste `L` sur le principe : la liste `L` est coupée en deux au milieu, chaque moitié est triée récursivement, puis on fusionne les listes grâce à la fonction `fusion`.

Indice : le milieu d'une liste de longueur est le quotient de la division euclidienne de la longueur de la liste par 2.

Solution:

```
def tri_fusion(L):
    n = len(L)
    if n<=1 :  # condition d'arrêt si L a moins d'un élément, elle est déjà triée
        return L
    else :
        L0 = tri_fusion( L[0:n//2] ) # tri récursif de la première moitié de L
        L1 =  tri_fusion( L[n//2:] ) # tri récursif deuxième moitié de L
        return fusion(L0,L1) # fusion des deux listes triées
```

Ce **tri n'est pas en place** (la liste `L` n'est pas modifiée, une autre est créée), de complexité moyenne $O(n\ln(n))$.

R5. Un autre tri récursif : le tri bulle. À chaque parcours de la liste, on fait monter la plus grande valeur de la partie qui reste à trier, au bout de la partie qui reste à trier.

Solution:

```
def tri_bulle(L:list)->list:
    n = len(L)
    if n<=1 :  # condition d'arrêt si L a moins d'un élément, elle est déjà triée
        return L
    else : # il faut parcourir la liste jusqu'au dernier rang, pour y placer la plus grande valeur de la liste L
        for i in range(0,len(L)-1) : # parcours de L ATTENTION à la valeur maximale de i dans le range vue le contenu de la boucle
            if L[i]>L[i+1]: # l'élément précédent est plus grand que le suivant
                L[i] , L[i+1] = L[i+1] , L[i] # permutation des éléments de rang i et i+1
        # à la fin de la boucle, le plus grand élément de L est arrivé au rang n-1 de L
        return tri_bulle(L[:len(L)-1]) + [ L[len(L)-1] ] # appel récursif sur la liste L privée de son dernier élément (qui est à la bonne place), avec concaténation de ce dernier élément.
```

C'est un **tri en place** (=modification de la liste `L`), de **complexité quadratique**.