Fonctions Récursives

Compétences

- Écrire un algorithme avec une méthode récursive
- Visualiser la notion d'environnement et de pile d'exécution.

Une série de petits exercices

Pour tester vos fonctions et mieux comprendre les appels récursifs il est conseillé d'utiliser Python tutor

1. Rendre récursive la fonction somme suivante :

```
1 def somme(L):
2     s = 0:
3     for valeur in L :
4         s += valeur
5     return s
```

2. Écrire une fonction qui prend en argument une chaîne de caractères et qui renvoie la chaîne écrite à l'envers.
3. Écrire une fonction qui prend en argument un nombre binaire sous la forme d'une chaîne de caractères et qui renvoie le nombre complémentaire : 0110 -> 1001
4. Écrire une fonction qui prend en arguments un tableau, sa taille et un indice et qui renvoie la somme des entiers positifs entre l'indice inclu et la fin du tableau.
   Le tableau `[2, 3, -1, 4]` avec l'indice 1 renvoie 7
5. Écrire une fonction qui prend en argument un tableau, sa taille et qui renvoie un tableau dont les éléments sont rangés dans l'ordre inverse.
6. Un nombre N est pair si $(N-1)$ est impair, et un nombre N est impair si $(N-1)$ est pair. Ecrire deux fonctions récursives croisées `pair(N)` et `impair(N)` permettant de savoir si un nombre N est pair ou impair.
7. Écrire une fonction qui prend en argument un tableau, sa taille et qui renvoie le maximum de ce tableau. Pensez à diviser pour régner.
8. Écrire une fonction qui prend en argument un tableau, sa taille et qui renvoie `True` si le tableau est trié et `False` sinon.

La suite de Fibonnaci

Considérons la suite numérique ainsi définie :

- $F_0 = 0$
- $F_1 = 1$
- $\forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n$

On a donc :
$F_2 = 0+1 = 1$
$F_3 = F_2 + F_1 = 1+1 = 2$,
$F_4 = F_3 + F_2 = 2+1 = 3, \ldots$

1. Écrire une fonction `fiboRecursive` qui calcule à l'aide d'une méthode recursive le n-ième terme de la suite de Fibonnaci.
2. Observer en détail la pile d'exécution lors du calcul de F(4) avec PythonTutor
3. Écrire une fonction `fiboIterative` qui calcule à l'aide d'une méthode itérative le n-ième terme de la suite de Fibonnaci.
4. Comparer grâce au module `time`(TP2) le temps moyen pris pour calculer le terme de rang $n = 20$ avec les deux fonctions `fiboIterative` et `fiboRecursive`

Exponentiation rapide

On se donne un scalaire x et un entier naturel n et on cherche à calculer x^n en utilisant que la multiplication et sans utiliser $x**n$ de Python.

- La première idée est d'utiliser la définition de la puissance entière qui est donnée par la relation de récurrence :
  $x^0 = 1$ et pour tout $m \in \mathbb{N}$, $x^{m+1} = x * x^m = x^m * x$
- La deuxième idée est d'utiliser le principe suivant :
  - Si n est pair alors $n^n = x^{n/2} * x^{n/2}$
  - Si n est impair alors $x^n = x^{(n-1)/2} * x^{(n-1)/2} * x$

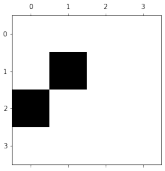
1. Écrire pour chaque méthode une fonction de paramètre n et qui renvoie la valeur de x^n
2. Comparez les temps de calcul des deux versions. Vous ferez cette comparaison pour la plus grande valeur de n possible. Comment s'explique la différence des deux temps de calcul?

Labyrinthe

Un labyrinthe peut-être représenté à l'aide d'une matrice carrée de la forme : $\mathscr{M}_{n,n}(\{0,1\})$ avec $n \in \mathbb{N}^*$. On décide que la valeur 1 indique un couloir et que la valeur 0 indique un mur. À partir d'une cellule donnée, nous sommes autorisés à passer aux cellules $(i+1, j)$ et $(i, j+1)$ uniquement

Exemple d'un labyrinthe de dimension (4×4)

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$



1. Tracer à la main tous les chemins possibles pour aller de la case $(0,0)$ à la case $(3,3)$
2. Écrire une fonction récursive permettant de calculer tous les chemins possibles en partant de la case $(0,0)$ pour atteindre la case $(n-1, n-1)$