2.4 Exercices

NSI Tle - JB Duthoit

⚠ Bien que très pratique, le ***slicing*** n'est pas au programme de NSI.
☛ Dans la suite, on pourra créer puis utiliser une fonction `extraction(string,i,j)` avec `string` une chaîne de caractères, et `i` et `j` deux entiers, avec `i<=j`. La fonction `extraction` renvoie l'extraction de la chaine de caractères `string` de l'indice `i` inclus à l'indice `j` non inclus.

2.4.1 Longueur d'une liste

Exercice 2.30
Imaginer un programme récursif qui retourne la longueur d'une liste.

2.4.2 Fibonacci

Exercice 2.31
Proposer une fonction récursive pour calculer le terme d'indice n de la suite de Fibonacci.
Les premiers termes sont 1, 1, 2, 3, 5, 8, 13, 21... Le termes suivant est calculé en ajoutant les deux derniers termes.
Dessiner le graphe des appels successifs de fib(4).

2.4.3 Palindrome

Exercice 2.32
Un palindrome est un mot dont les lettres, lues de droites à gauche sont les mêmes que celles lues de gauche à droite.
Les mots radar, elle, été, kayak sont des palindromes.
Proposez un programme récursif qui teste si un mot est un palindrome.

2.4.4 Somme d'une liste

Exercice 2.33
Proposer une fonction récursive somme qui calcule la somme des éléments d'une liste d'entiers passée en paramètre.
On supposera que la somme d'une liste vide est 0.

2.4.5 Autres exercices

Exercice 2.34
on considère la fonction myst(l) qui prend une liste en entrée et définie récursivement par :

```
def myst (l):
        if l == []:
                return 0
        else :
```

```
                return 1 + myst (l [1:])
```

Que fait cette fonction ?

Exercice 2.35
Proposer une fonction nommée *retourner* paramétrée par une chaîne de caractère et qui retourne la même chaîne écrite de droite à gauche.

Exercice 2.36
Rendre récursif la fonction suivante :

```
def somme(L):
    s=0 :
    for val in L :
        s+=val
    return s
```

Exercice 2.37
On suppose défini le dictionnaire :

```
VALEUR_ROMAIN = { 'M' : 1000, 'D' : 500, 'C' : 100, 'L' : 50,
                  'X' : 10, 'V' : 5, 'I' : 1}
```

Réalisez une fonction récursive $romain_to_arabe$ qui prend en paramètre une chaîne de caractères représentant un « nombre romain » et dont le résultat est l'entier correspondant.

```
>>> romain_to_arabe('X')
10
>>> romain_to_arabe('XCI')
91
>>> romain_to_arabe('MMXIX')
2019
```

NB Il est nécessaire de prendre en compte le cas où la valeur correspondante au second caractère est supérieure à celle du premier !

Exercice 2.38
Réalisez une version récursive du tri par insertion vu en première

Exercice 2.39
Soit un la suite d'entiers définie par $u_{n+1} = u_n/2$ si un est pair, et $u_{n+1} = 3 \times u_n + 1$ sinon. avec u_0 un entier quelconque plus grand que 1. Écrire une fonction récursive $syracuse(u_n)$ qui affiche les valeurs successives de la suite un tant que un est plus grand que 1.
La conjecture de Syracuse affirme que, quelle que soit la valeur de u_0, il existe un indice n dans la suite tel que $u_n = 1$. Cette conjecture défie toujours les mathématiciens !

Exercice 2.40
Ecrire une fonction récursive $nombre\_de\_chiffres(n)$ qui prend un entier positif ou nul n en argument et renvoie son nombre de chiffres.
Par exemple, $nombre\_de\_chiffres(34126)$ doit renvoyer 5.

Exercice 2.41

*** Dans le cadre d'un championnat sportif (ou autre) on dispose de la liste de tous les joueurs (ou équipes) concernés. On souhaite organiser la liste de toutes les rencontres possibles entre ces joueurs. Chaque joueur devant rencontrer tous les autres une et une seule fois.

On considère que chaque joueur est identifié par un nombre (qui peut par exemple correspondre à une clef dans une table qui permet d'accéder aux informations détaillées sur le joueur). Une liste de joueurs est donc en fait la liste des nombres associés à ces joueurs.

Une rencontre est représentée par un couple (tuple) dont les deux composantes sont les numéros des deux joueurs impliqués.

Donnez et codez un algorithme récursif qui produit, à partir d'une liste de joueurs, la liste de toutes les parties possibles entre ces joueurs.

```
>>> rencontres([1,2,3,4])
[ (1,2) , (1,3) , (1,4), (2,3), (2,4), (3,4) ]
```

Exercice 2.42

Proposer une fonction récursive `add(a,b)` de calcul de la somme de deux entiers naturels a et b en supposant que les seules opérations de base sont :

- L'ajout de 1 à un entier
- Le retrait de 1 à un entier
- les comparaisons de 0 à un entier (=,>,<)

Exercice 2.43

Proposer une fonction récursive `mult(a,b)` de calcul du produit de deux entiers naturels a et b en supposant que les seules opérations de base sont :

- La somme de deux entiers
- Le retrait de 1 à un entier
- les comparaisons de 0 à un entier (=)