2.2 Exercices

NSI terminale - JB Duthoit

2.2.1 Pour commencer...

Exercice 2.2
Construire une fonction récursive revoyant la puissance d'un entier naturel
Soit a un entier non nul et n un entier naturel.
En sachant que $a^0 = 1$ et $a^n = a^{n-1} \times a$, construire la fonction récursive `puissance(a,n)` qui prend comme paramètre `a` et `n` et qui retourne le résultat de a^n.

Exercice 2.3
Savoir calculer la somme des premiers entiers consécutifs.
Construire une fonction récursive `somme(n)` qui permet de calculer :

$$0 + 1 + 2 + 3 + ... + n$$

Exercice 2.4
Proposer une fonction récursive `fibo(n)` pour calculer le terme d'indice `n` de la suite de Fibonacci.
Les premiers termes sont 0, 1, 1, 2, 3, 5, 8, 13, 21...
On considère que `fibo(0)=0` et que `fibo(1) = 1`.
Pour $n \geq 2$, un terme est obtenu en additionnant les deux précédents.
Dessiner le graphe des appels successifs de fib(4).

2.2.2 Avec un paramètre par défaut

Exercice 2.5
Dans cet exercice, on utilisera un paramètre par défaut.
Le début du code peut être :

```
def somme(lst, j = None):
  if j is None:
    j = len(lst) -1
```

Exercice 2.6
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 une fonction récursive `pal(mot)` qui teste si une chaîne de caractères `mot` est un palindrome ou non. On imaginera deux solutions :

1. En utilisant le fonction `extraction(chaine,i,j)`.
2. En utilisant un paramètre par défaut. Voici le début de la fonction :

```
def pal(mot,i = 0, j = None):
```

```
  if j is None:
    j = len(mot) - 1
  if j <= i:
    return True
```

Exercice 2.7
Proposer une fonction nommée retourner paramétrée par une chaîne de caractère et qui retourne la même chaîne écrite de droite à gauche. On utilisera ici un paramètre par défaut.
Le début de la fonction peut être :

```
def retourner(mot, n = 0):
  if n = len(mot):
    return ''
  return ...
```

2.2.3 Autres exercices

Exercice 2.8
Rendre récursif la fonction suivante :

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

Exercice 2.9
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 2.10
*** Réalisez une version récursive du tri par insertion vu en première

Exercice 2.11
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 u_n tant que u_n 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 2.12
Écrire 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 2.13
*** 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 2.14
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 possibles sont :

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

Exercice 2.15
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 (=)

Exercice 2.16

Ecrire une fonction récursive `binaire(N)` où :

- N est un entier positif, donné en base 10.
- La fonction renvoie une chaine de caractères correspondant à la représentation binaire du nombre N

***

Exercice 2.17

On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction `echange` qui permet d'échanger dans une liste `lst` les éléments d'indice `i1` et `i2`. Expliquer pourquoi le code Python ci-dessous ne réalise pas cet échange et en proposer une modification.

```
def echange(lst, i1, i2):
    lst[i2] = lst[i1]
    lst[i1] = lst[i2]
```

2. La documentation du module `random` de Python donne : « `randint(a, b)` renvoie un entier aléatoire N tel que $a \leq N \leq b$ ». Parmi les valeurs ci-dessous, lesquelles peuvent être renvoyées par l'appel `randint(0, 10)` ?

0 1 3.5 9 10 11

3. Le mélange de **Fisher–Yates** ci-dessous permute aléatoirement les éléments d'une liste (version récursive).

```
from random import randint

def melange(lst, ind):
    print(lst)
    if ind > 0:
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

   a) Expliquer pourquoi la fonction `melange` se termine toujours.

   b) Lors de l'appel initial, le paramètre `ind` doit être égal au plus grand indice de la liste `lst`. Pour une liste de longueur n, quel est le **nombre d'appels récursifs** effectués (sans compter l'appel initial) ?

   c) On considère le script :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

   On suppose que les valeurs successivement renvoyées par `randint` sont `2`, `1`, `2` et `0`. Les deux premiers affichages produits par `print(lst)` sont :

$$[0, 1, 2, 3, 4] \quad \text{puis} \quad [0, 1, 4, 3, 2].$$

   Donner les affichages **suivants** produits par `melange`.

   d) Proposer une version **itérative** (non récursive) du mélange de Fisher–Yates.

Exercice type BAC 2.18

(Sujet 0.b de 2022)

Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.

1. a) Expliquer en quelques mots ce qu'est une fonction récursive.

   b) On considère la fonction Python suivante :

```
def compte_rebours(n):
    if n >= 0:
        print(n)
        compte_rebours(n-1)
```

   L'appel compte_rebours(3) affiche successivement les nombres 3, 2, 1 et 0. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 0.

2. En mathématiques, la factorielle d'un entier naturel n est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Par convention, la factorielle de 0 est 1. Par exemple :

   - la factorielle de 1 est 1
   - la factorielle de 2 est 2 x 1 =2
   - la factorielle de 3 est 3 x 2 x 1 = 6
   - la factorielle de 4 est 4 x 3 x 2 x 1 = 24...

   Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive fact renvoie la factorielle de l'entier passé en paramètre de cette fonction.

   Exemple : fact(4) renvoie 24.

```
def fact(n):
    if n == 0:
      return ...
    else:
      return ...
```

3. La fonction somme_entiers_rec ci-dessous permet de calculer la somme des entiers, de 0 à l'entier naturel n passé en paramètre.

   Par exemple :

   - Pour n = 0, la fonction renvoie la valeur 0.
   - Pour n = 1, la fonction renvoie la valeur 0 + 1 = 1.
   - ...
   - Pour n = 4, la fonction renvoie la valeur 0 + 1 + 2 + 3 + 4 = 10.

```
def somme_entiers_rec(n):
    """ Permet de calculer la somme des entiers,
    de 0 à l'entier naturel n """
    if n == 0:
        return 0
    else:
        print(n)  # pour vérification
        return n + somme_entiers_rec(n - 1)
```

L'instruction `print(n)` de la ligne 7 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en œuvre au niveau des appels récursifs.

a) Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante :

```
res = somme_entiers_rec(3)
```

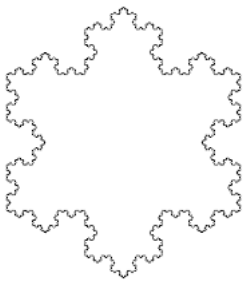
b) Quelle valeur sera alors affectée à la variable res ?

4. Écrire en Python une fonction somme_entiers non récursive : cette fonction devra prendre en argument un entier naturel n et renvoyer la somme des entiers de 0 à n compris. Elle devra donc renvoyer le même résultat que la fonction somme_entiers_rec définie à la question 3.
Exemple : somme_entiers(4) renvoie 10.

Exercice 2.19

Savoir construire un programme récursif pour dessiner (plus difficile)

Il s'agit ici de construire un programme qui permet de dessiner le flocon de Von Koch :



Il faut commencer à vous familiariser avec le module turtle de Python, qui permet de dessiner facilement.

Tester et analyser ce petit programme :

```
from turtle import *
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

```
forward(100)
right(30)
forward(100)
right(120)
forward(100)
```

Puis il faut approfondir le sujet avec : Wikibook sur le module turtle Python
Et enfin se familiariser avec le flocon de Von Koch en visionnant cette vidéo : Vidéo sur le flocon