

2.2

Exercices

2.2.1 Exercices sans paramètre par défaut

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 `fonction(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.

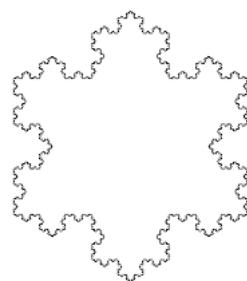
Construire une fonction récursive `somme(n)` qui permet de calculer :

$$0 + 1 + 2 + 3 + \dots + n$$

Exercice 2.4

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

Exercice 2.5

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 $\text{fibo}(0)=0$ et que $\text{fibo}(1) = 1$.

Pour $n \geq 2$, un terme est obtenu en additionnant les deux précédents.

Dessiner le graphe des appels successifs de `fib(4)`.

⚠ Bien que très pratique, le **slicing** n'est pas au programme de NSI.

Voici les différentes possibilités pour y remédier :

- ☛ 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 chaîne de caractères `string` de l'indice `i` inclus à l'indice `j` non inclus.
- ☛ On peut aussi utiliser une compréhension de liste
- ☛ Afin d'éviter de créer des nouveaux tableaux, on peut aussi créer une fonction récursive avec un paramètre en plus (ce paramètre peut correspondre par exemple à un indice). Cette méthode est en général la plus performante.

2.2.2 Exercices préliminaires

Exercice 2.6

Construire une fonction `extraction(chaine,i,j)` avec `chaine` une chaîne de caractères, et `i` et `j` deux entiers, avec `i<=j`.

La fonction `extraction` renvoie l'extraction de la chaîne de caractères `string` de l'indice `i` inclus à l'indice `j` non inclus.

Par exemple, `extraction('voiture',2,4)` va renvoyer 'it'

Exercice 2.7

Construire une fonction `extractionliste(lst,i,j)` avec `lst` un tableau python, et `i` et `j` deux entiers, avec `i<=j`.

La fonction `extractionliste` renvoie l'extraction du tableau `lst` de l'indice `i` inclus à l'indice `j` non inclus.

Par exemple, `extraction([1,2,3,4,5,6],2,5)` va renvoyer [3,4,5].

2.2.3 Exercices

Exercice 2.8

Créer une fonction récursive `long(t)` avec comme paramètre `t` un tableau dynamique python et qui renvoie la longueur du tableau `t`.

On utilisera la fonction `extraction(chaine,i,j)`.

Exercice 2.9

Créer une fonction récursive `long(t, indice = None)` avec comme paramètre `t` un tableau dynamique python, `j` qui correspond à l'indice jusque où l'on considère le tableau et qui renvoie la longueur du tableau `t`.

Le début du code peut-être :

```
def long(lst, indice = None):
    if indice == None:
        indice = len(mot) - 1
```

Exercice 2.10

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 la 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 2.11

SAVOIR CALCULER LE NOMBRE D'OCCURRENCES D'UN CARACTÈRE DANS UNE CHAÎNE.

Construire une fonction récursive `occurrenec(chaine,caractere)` qui prend en paramètre deux chaînes de caractères `chaine` et `caractere`. La fonction renvoie le nombre d'occurrence(s) de `caractere` dans `chaine`.

- Le nombre d'occurrences de `caractere` dans `chaine` est 0 si `chaine` est vide.
- Si `caractere` est le premier caractère de `chaine`, on ajoute 1 au nombre d'occurrences de `caractere` dans les autres caractères de `chaine`
- Sinon, il s'agit du nombre d'occurrences de `caractere` dans les autres caractères de `chaine`.

Avant de réaliser cet exercice, on créera une fonction `sansdebut(chaine)` avec comme paramètre `chaine` une chaîne de caractère, et qui renvoie la même chaîne de caractère sans le premier caractère. Ainsi `sansdebut("voiture")` donnera "oiture". Pour créer cette fonction `sansdebut(chaine)`, on s'interdira d'utiliser le slicing.

Exercice 2.12

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.

On imaginera deux solutions :

1. Une première utilisant la fonction `extractionlst`
2. Et une autre utilisant des paramètres par défaut. Le début du code peut être :

```
def somme(lst, j = None):
```

```

if j is None:
    j = len(mot) - 1

```

Exercice 2.13

on considère la fonction `myst(l)` qui prend une liste en entrée et définit récursivement par :

```

def myst (l):
    if l == []:
        return 0
    else :
        return 1 + myst (l [1:])

```

Que fait cette fonction ?

Exercice 2.14

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 ...

```

Exercice 2.15

Rendre récursif la fonction suivante :

```

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

```

Exercice 2.16

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.

```

>>> roman_to_arabe('X')
10
>>> roman_to_arabe('XCI')
91
>>> roman_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 2.17

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

Exercice 2.18

Soit un la suite d'entiers définie par $u_{n+1} = u_n/2$ si u_n 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 2.19

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 2.20

*** 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 clé 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 2.21

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 2.22

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 2.23**

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

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

 **Exercice type BAC 2.24**

| Exercice 2 de l'épreuve "sujet 0-b" de novembre 2022