

## Série 7

Sauf si spécifié autrement, tous les algorithmes demandés sont à écrire sous forme d'une fonction Python.

### Exercice 1

On donne l'algorithme suivant :

```
def f(a, b):  
    '''  
    Input: a, b entiers, a >= b > 0  
    '''  
    r = a  
    while r >= b:  
        r -= b  
  
    return r
```

- (a) Donner la valeur de  $f(3, 2)$ ,  $f(20, 16)$ ,  $f(20, 5)$ .
- (b) Que calcule cet algorithme? Prouver sa correctitude.

### Exercice 2

Formulez et prouvez un invariant de boucle pour prouver que la fonction `somme` ci-dessous calcule bien, pour un entier  $n \geq 1$  en entrée, la somme  $1 + 2 + \dots + n$ .

```
def somme(n):  
    s = 0  
    for i in range(1, n+1):  
        s += i  
    return s
```

### Exercice 3

- (a) Ecrivez un algorithme `countdown` récursif (sans boucle) qui prend un entier positif  $n$  en entrée et affiche les nombres de  $n$  à  $0$  par ordre décroissant. Par exemple, `countdown(3)` doit afficher

```
3  
2  
1  
0
```

N'oubliez pas de tester votre algorithme pour quelques petites valeurs de  $n$ .

Que se passe-t-il si vous appelez votre algorithme avec l'argument  $n = -1$  ?

- (b) Ecrivez un algorithme itératif qui produit le même affichage.

#### Exercice 4

Ecrivez un algorithme récursif (sans boucle) qui prend en entrée une liste  $L$  non vide de nombres et retourne le maximum de  $L$ .

N'oubliez pas de tester votre algorithme sur quelques exemples de  $L$ .

Que se passe-t-il si vous appelez votre algorithme avec une liste vide ?

#### Exercice 5

Vous avez vu au cours des algorithmes récursifs pour calculer la factorielle d'un nombre  $n$  et le  $n$ -ième nombre de Fibonacci. Dans cet exercice, vous allez implémenter des versions itératives (avec une boucle) de ces algorithmes.

- (a) Ecrivez un algorithme `fact_iter` qui prend en entrée un entier  $n \geq 0$  et calcule itérativement (avec une boucle, sans appels récursifs) la factorielle de  $n$ .
- (b) Prouvez la correctitude de cet algorithme.
- (c) Ecrivez un algorithme `fib_iter` qui prend en entrée un entier  $n \geq 0$  et calcule itérativement (avec une boucle, sans appels récursifs) le  $n$ -ième nombre de Fibonacci. Regardez en bas de page pour un **indice**.<sup>1</sup>
- (d) Prouvez la correctitude de cet algorithme.

#### Exercice 6

Ecrivez une version récursive (sans boucle `while`) de l'algorithme d'Euclide vu en cours.

#### Exercice 7

Pour une chaîne de caractères  $s$ , on définit une **sous-chaîne** de  $s$  (dans le cadre de cet exercice) comme un sous-ensemble des caractères de  $s$ , apparaissant dans le même ordre que dans  $s$ . Par exemple, si  $s = \text{"abac"}$ , alors  $\text{"b"}$ ,  $\text{"aa"}$ ,  $\text{"bc"}$  et la chaîne vide sont des sous-chaînes de  $s$ , mais  $\text{"ca"}$  et  $\text{"cc"}$  ne le sont pas.

---

1. A travers les itérations de la boucle, vous devez vous rappeler de deux valeurs : les deux derniers nombres de Fibonacci calculés.

On veut écrire un algorithme récursif `sous_chânes` qui prend en entrée une chaîne de caractères `s` et produit une liste contenant toutes les sous-chaînes contenues dans `s` (en permettant les répétitions, et dans un ordre quelconque).

Par exemple, `sous_chânes("abc")` doit retourner (à l'ordre près)

`['', 'c', 'b', 'bc', 'a', 'ac', 'ab', 'abc']`

et `sous_chânes("aa")` doit retourner (à l'ordre près)

`['', 'a', 'a', 'aa']`.

- (a) Pour une chaîne de caractères non vide `s`, supposez que vous avez à disposition la liste `L` des sous-chaînes de `s[1:]`. À partir de cette liste `L`, comment forme-t-on la liste des sous-chaînes de `s` ?

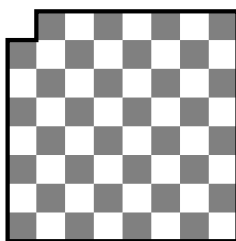
(Par exemple, pour la chaîne `s = "abc"`, on a `s[1:] = "bc"` et la liste de sous-chaînes de `s[1:]` est `L = ['', 'c', 'b', 'bc']`.)

Déduisez-en le ou les appels récursifs de l'algorithme.

- (b) À quelle chaîne de caractères en entrée correspond le cas de base ? Que faut-il sortir dans ce cas ?
- (c) Donnez l'algorithme `sous_chânes`.

### Exercice 8 (facultatif)

- (a) On dispose d'un échiquier où il manque un coin, comme dans la figure ci-dessous, et de dominos pouvant chacun couvrir deux cases adjacentes d'un échiquier. Peut-on couvrir cet échiquier avec ces dominos entièrement et exactement (sans aucun domino qui dépasse) ?



- (b) Même question pour l'échiquier ci-dessous, avec deux coins manquants.

