Série 7

Sauf si spécifié autrement, tous les algorithmes demandés sont à écrire sous forme d'une fonction Python.
Les exercices précédés d'une astérisque sont optionnels (mais pas nécessairement difficiles).

---

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?

   *(c) Prouver sa correctitude.

2. (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(4)` doit afficher

```
4
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.

3. 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?

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

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 $\texttt{n} \geq \texttt{0}$ 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 $\texttt{n} \geq \texttt{0}$ et calcule itérativement (avec une boucle, sans appels récursifs) le `n`-ième nombre de Fibonacci. Regardez en bas de page pour un **indice**[1].

   *(d) Prouvez la correctitude de cet algorithme.

6. 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 = "abac"`, alors `"b"`, `"aa"`, `"bc"` et la chaîne vide sont des sous-chaînes de `s`, mais `"ca"` et `"cc"` ne le sont pas.

   On veut écrire un algorithme récursif `sous_chaînes` 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_chaînes("abc")` doit retourner (à l'ordre près)
   `['', 'c', 'b', 'bc', 'a', 'ac', 'ab', 'abc']`
   et `sous_chaînes("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:]`. A partir de cette liste `L`, comment forme-t-on la liste des sous-chaînes de `s`?

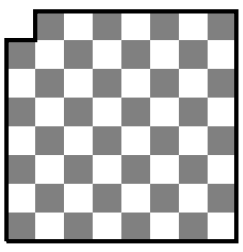
   (Par exemple, pour la chaîne `s = "abc"`, on a `s[1:] = "bc"` et la liste de sous-chaînes de `s[1:]` est `L = ['', 'c', 'b', 'bc']`.)

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

   (b) A quelle chaîne de caractères en entrée correspond le cas de base? Que faut-il sortir dans ce cas?

   (c) Donnez l'algorithme `sous_chaines`.

* 7. (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)?



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

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