Laboratoire # 4
Marchons plus loin...

Définition et réutilisation de fonctions

Semaine 4

Ce quatrième laboratoire a pour but de revenir sur la notion de fonction, et de résoudre un nouveau problème en utilisant les outil (fonctions) que nous avons déjà ! Réalisez ce laboratoire **sans regarder les fonctions définies dans les notes de cours** : vous devriez y arriver ! Nous vous demanderons de programmer des fonctions dont la solution est dans les notes de cours, mais essayez de vous rappeler de la manière de programmer celles-ci en y réfléchissant.

Note très importante : comme toujours, **ne regardez pas la solution avant d'avoir résolu le problème**. Ce cours doit vous apprendre à réfléchir autrement, et vous atteindrez cet objectif qu'avec de la pratique.

1 Réalisation de la fonction `diviseur(a, b)`

Sans regarder la solution, réalisez la fonction `diviseur(a, b)`, qui détermine si un entier `b` est un diviseur d'un entier `a`. Documentez votre fonction à l'aide du *Google Style Python Docstrings*, présenté brièvement dans les notes de cours.

2 Réalisation de la fonction `premier(n)`

Sans regarder la solution, réalisez la fonction `premier(n)`, qui détermine si un nombre entier `n` est premier. N'oubliez pas de réutiliser la fonction `diviseur(a, b)` définie plus haut.

Améliorez la fonction en vous demandant s'il est vraiment nécessaire de tester tous les entiers entre `2` et `n-1`. On peut s'arrêter avant ?

Documentez votre fonction !

3 Factorisation d'un nombre en deux nombres premiers

Soit deux grands nombres premiers donnés, il est facile d'en obtenir le produit. Par contre, il est beaucoup plus difficile de trouver les facteurs premiers de celui-ci. C'est ce que l'on appelle une *fonction trappe*. Ceci s'applique pour les systèmes modernes en *cryptologie*. Si une méthode rapide était trouvée pour résoudre le problème de la factorisation des nombres entiers, alors plusieurs systèmes cryptologiques importants seraient cassés, incluant *l'algorithme à clé publique RSA*. La mise au point d'un ordinateur quantique est l'une de ces méthodes.[1]

Écrivez une fonction `trouver_deux_facteurs_premiers(n)` qui affiche à l'écran, lorsqu'il y a une solution, deux *facteurs premiers* de l'entier `n`. Un facteur de `n` est simplement un diviseur de `n`. Un facteur premier est donc un nombre qui est **à la fois** un diviseur de `n` et un nombre premier.

Si deux facteurs ne sont pas trouvés, la fonction affiche que le nombre n ne peut pas être factorisé par deux nombres premiers. *La fonction ne retourne rien*, elle ne fait qu'afficher ses informations à l'écran.

La première tâche consiste à trouver un nombre `i` qui est à la fois un diviseur de `n` et un nombre premier. Une fois un tel `i` trouvé, il reste à valider que le second facteur associé (nommons le `j`) est également premier.

On cherche donc deux nombres i et j tels que i est premier, j est premier, et $i \times j = n$.

Notez que cet exercice comporte des similarités avec votre TP1, mais l'algorithme demandé est bien différent de ce que vous avez programmé au TP1. Tenter de récupérer la solution de cet exercice pour la transformer en une solution pour le TP1 est une tâche beaucoup plus difficile que de solutionner le TP1 directement. :-)

Documentez votre fonction, et testez-la avec plusieurs nombres pour la valider.

[1] Source : Wikipédia (http://fr.wikipedia.org/wiki/D%C3%A9composition_en_produit_de_facteurs_premiers)