| Université de Sfax | Année Universitaire : 2016-2017 |
|---|---|
| Institut Préparatoire aux Etudes d'Ingénieurs de Sfax | Section : BG1<br>Matière : Informatique<br>Durée : 2h<br>Mai 2017 |

# EXAMEN DE FIN DE SEMESTRE 2

## Exercice 1 (5 points)

La conjecture de Goldbach (formulée en 1742) dit que "tout nombre pair supérieur à 3 est la somme de deux nombres premiers".

L'objectif de cet exercice est de déterminer, à partir d'un nombre pair **n**, les entiers premiers **a** et **b** vérifiant la conjecture de GoldBach. Pour cela, on se propose de générer tous les couples **(a,b)** tels que **a + b = n**, en faisant varier **a** de **1** à **n/2**, puis vérifier pour chacun des couples **(a,b)** si **a** et **b** sont simultanément premiers.

**Exemple** :

Pour **n = 10**, on génère les couples suivants : **(1,9)**, **(2,8)**, **(3,7)**, **(4,6)**, **(5,5)**.

La conjecture de Goldbach est alors vérifiée pour les couples **(3,7)** et **(5,5)**.

On vous demande d'écrire les fonctions Python suivantes :

1. **saisie()**, sans paramètres et qui retourne le nombre entier pair **N** saisi au clavier et compris entre 4 et 1000 ;
2. **checkPrime(N)**, qui reçoit un entier **N** et retourne un résultat booléen : **True** si **N** est un nombre premier (les diviseurs de **N** sont seulement 1 et lui-même), **False**, sinon.
3. **genCouples(N)**, qui reçoit un entier **N**, génère les $\ell$ couples **(a,b)** tels que **a + b = N** et les retourne dans un tableau **T** à deux dimensions ($\ell$ lignes, 2 colonnes).
   **Indication** : Pour avoir un tableau à deux dimensions ($\ell$ lignes, 2 colonnes), vous pouvez convertir la liste contenant les $\ell$ couples **(a,b)** en un tableau avec la fonction **array**, du module **numpy**, qui prend en argument une liste et retourne un tableau.
4. **goldCouples(T)**, qui reçoit un tableau **T** à deux dimensions ($\ell$ lignes, 2 colonnes) et affiche les couples **(a,b)** vérifiant la conjecture de GoldBach ainsi que leur nombre (`for t in T` : permet le parcours de T, couple par couple). Pour l'exemple précédent, la fonction affichera les couples **(3,7)** et **(5,5)**.

Ecrire un programme principal qui permet de saisir un entier pair **NB** compris entre 4 et 1000 et d'afficher tous les couples de la conjecture de GoldBach en utilisant les fonctions précédentes.

## Exercice 2 (5 points)

La suite de *Fibonacci* est définie par la fonction $f$ :

$$\begin{cases} f(0) = 1 \; et \; f(1) = 1 \\ f(n) = f(n-1) + f(n-2) \; ; \; pour \; tout \; n \geq 2 \end{cases}$$

Ecrire un programme Python permettant de :

1. Définir la fonction récursive ***f(n)*** calculant le terme d'indice **n** de cette suite.
2. Demander à l'utilisateur de saisir un entier strictement positif **N** correspondant au nombre de termes de la suite qu'on va calculer.
3. Construire la liste **lx** formée par les **N** nombres entiers compris entre **0** à **N-1**, puis, la liste **ly** contenant les valeurs des **N** premiers termes de la suite calculés avec la fonction ***f***.
   **Exemple** : Pour **n** = 6, **lx** = [0, 1, 2, 3, 4, 5] et **ly** = [1, 1, 2, 3, 5, 8].
4. Tracer la courbe représentative de la fonction ***f***.
5. Ecrire dans un fichier **"fibo.txt"** les valeurs des termes de la suite en utilisant les deux listes précédentes. Chaque ligne du fichier contiendra un terme (La 1ère ligne contient : $f(0)=1$, la 2ème ligne contient $f(1)=1$ et ainsi de suite).

## Problème (10 points)

### I. Génération aléatoire d'une séquence d'ADN

On se propose de construire une chaîne de caractères correspondante à une séquence d'ADN de longueur $\ell$ spécifiée par l'utilisateur. Cette séquence comporte uniquement les caractères "A", "T", "C" et "G".

Ecrire les fonctions Python suivantes :

I.1. **saisie()**, sans paramètres et qui retourne un entier strictement positif **n** saisi par l'utilisateur (ce nombre **n** correspond à la longueur de la séquence d'ADN à générer).

I.2. **generTab(n)** qui prend en argument un entier strictement positif **n** et renvoie un tableau de type **numpy.ndarray** en Python. Ce tableau est rempli par **n** nombres entiers entre **0** et **3** étant générés de façon aléatoire. En python, on a dans le module **random**, la fonction **random.randint(a,b)** qui permet de générer aléatoirement un entier entre **a** et **b** (b inclus).

I.3. **construiADN(T)** qui prend en argument un tableau **T** et retourne une chaîne de caractères ADN de même taille. Cette chaîne est formée par les correspondances entre les éléments du tableau **T** et les indices des lettres "A", "T", "G" et "C" dans la liste définie par **L = ["A", "T", "G" , "C"]**.
Par exemple, si le tableau **T** = [**0**, 1, 3, 0, 2, 1], alors **ADN** = "ATCAGT".

L[0]

## II. Enregistrement d'une séquence d'ADN dans un fichier

On se propose maintenant d'enregistrer une séquence d'ADN générée aléatoirement dans un fichier texte "séquenceADN.txt".

II.1. Ecrire les instructions Python faisant appels aux fonctions précédentes et permettant de :

- saisir un entier strictement positif $\ell$ correspondant à la longueur de la séquence d'ADN à générer ;
- Générer un tableau **T** de taille $\ell$ et rempli par des nombres entiers entre 0 et 3 étant générés de façon aléatoire ;
- Construire une chaîne de caractères **ADN** de longueur $\ell$ avec le tableau **T** généré précédemment ;

II.2. Enregistrer, dans le fichier "séquenceADN.txt", la chaîne de caractères **ADN** produite.

## III. Manipulation du fichier "séquenceADN.txt"

Maintenant que l'on dispose du fichier "séquenceADN.txt" contenant une chaîne de caractères correspondante à une séquence d'ADN générée de façon aléatoire, on vous demande d'écrire les fonctions Python suivantes :

III.1. **trouve(motif)** qui teste l'existence, d'une chaîne de caractères **motif**, donnée en paramètre, dans le fichier "séquenceADN.txt" (il est vérifié que **motif** ne contient que des "A", "G", "C" ou "T"). Cette fonction renvoie un résultat booléen : **True**, si le **motif** apparaît au moins une fois dans le fichier "séquenceADN.txt", **False**, sinon.

III.2. **Transcription()** qui ajoute au fichier "séquenceADN.txt" (après retour à la ligne) une séquence d'ARN : chaîne de caractères obtenue en modifiant les "T" en "U" dans la séquence d'ADN du fichier "séquenceADN.txt". La fonction **replace(char1,char2)** appliquée sur une chaîne de caractères retourne la chaîne modifiée en remplaçant `<char1>` par `<char2>`.

III.3. **Complémentaire()** qui ajoute au fichier "séquenceADN.txt" (après retour à la ligne) la séquence d'ADN obtenue en remplaçant "A" par "T", "C" par "G", "G" par "C" et "T" par "A", dans la **<u>première</u>** séquence du fichier (**<u>Attention</u>** : le fichier peut contenir plus qu'une séquence (ADN et ARN) étant séparées par " \n ").

III.4. **pourcentGC()** qui retourne le pourcentage en GC dans la **<u>première</u>** séquence d'ADN enregistrée dans le fichier "séquenceADN.txt". Il faut compter le nombre total de "G" et "C" dans les occurrences de "GC" présentes dans la séquence d'ADN, puis le diviser par la taille de toute la séquence.

Par exemple, si la séquence d'ADN est "ATGCTGCAAT", alors :

- le nombre d'occurrences de "GC" = **2**,
- le nombre total de "G" et "C" = **2***2= 4,
- et le pourcentage en GC= (4/10)*100 → 40 %