Concours Blanc : TP d'Informatique

PTSI Lycée Eiffel

6 juin 2019

L'objectif de ce TP est de programmer différentes méthodes de calcul des termes de la suite de Fibonacci. Il s'agit bien entendu purement d'une épreuve de programmation, les rares résultats mathématiques utiles seront donnés dans l'énoncé et ne seront absolument pas à démontrer. Les seules fonctions et méthodes sur les listes dont l'utilisation est autorisée au sein de vos programmes sont **len**, **append**, **del** et **remove**. En particulier, l'emploi de **sum**, **max**, **min** ou **sort** est interdit (mais vous pouvez bien sûr reprogrammer de telles fonctions si vous en avez besoin). Les candidats ont bien entendu le droit d'écrire des fonctions intermédiaires supplémentaires en plus de celles demandées explicitement dans le sujet s'ils estiment que celà peut améliorer la lisibilité de leur copie. Il est par ailleurs fortement conseillé de commenter les programmes contenant plus de quelques lignes de code. On rappelle que les commentaires en Python sont introduits par le caractère ♯.

I. Exploitation de la relation de récurrence.

On rappelle que la suite de Fibonacci est la suite récurrente double (u_n) définie par les données suivantes : $u_0 = 0$, $u_1 = 1$ et $\forall n \in \mathbb{N}$, $u_{n+2} = u_{n+1} + u_n$. Pour que vous puissiez vérifier la validité de certains de vos programmes, on donne la liste des 11 premiers termes de la suite (jusqu'à u_{10} inclus) : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ainsi que les valeurs $u_{20} = 6\ 765$, $u_{30} = 832\ 040$ et $u_{100} = 354\ 224\ 848\ 179\ 261\ 915\ 075$.

1. Écrire une fonction Python **fibo(n)** qui prend en argument un entier naturel n, et qui renvoie la valeur de u_n. Dans cette première version, on conseille d'utiliser deux variables réelles u et v pour effectuer les calculs intermédiaires (le recours à un algorithme récursif est autorisé mais fortement déconseillé ; en tout cas votre programme doit pouvoir calculer instantanément u_{1000} par exemple).
2. Écrire une fonction Python **fiboliste(n)** qui prend en argument un entier naturel n, et qui renvoie la liste de tous les termes de la suite, jusqu'à u_n inclus. On essaiera pour ce programme de ne pas utiliser d'autres variables qu'une variable de type liste.
3. On peut démontrer que les termes de la suite de Fibonacci vérifient pour tout entier naturel n l'égalité $u_{2n+1} = u_n^2 + u_{n+1}^2$. Écrire une fonction Python **verifiecarres(n)** qui vérifie la propriété précédente pour tous les entiers inférieurs ou égaux à n. On pourra bien sûr exploiter par exemple le programme écrit à la question précédente (pour une valeur de n bien choisie). La fonction verifiecarres renverra un booléen (True ou False, mais il faudra s'inquiéter si la fonction renvoie autre chose que True en pratique, les mathématiciens n'ayant pas l'habitude de démontrer n'importe quoi).
4. Écrire une fonction Python **premierplusgrand(p)** qui prend en argument un entier naturel p, et qui renvoie la valeur du premier terme de la suite de Fibonacci ayant une valeur supérieure ou égale à p (ainsi, premierplusgrand(1000000) devrait renvoyer 1 346 269).
5. Modifier le programme précedent pour écrire une fonction **plusgrandavecindice(p)** qui renvoie non seulement la valeur du premier terme supérieur à p, mais également son indice dans la suite.

6. Écrire une fonction Python **chiffres(n)** prenant en argument un entier naturel n et renvoyant la liste des chiffres apparaissant dans l'écriture décimale de n, rangée par ordre croissant (ainsi, chiffres(709641516) devrait renvoyer [0,1,4,5,6,7,9]). On évitera bien entendu de faire figurer plusieurs fois dans la liste les chiffres apparaissant plus d'une fois dans l'écriture de n. Aucune méthode n'est imposée pour cette question, les seules restrictions en vigueur étant celles imposées dans le préambule de l'énoncé.
7. Écrire une fonction Python **touschiffres()** ne prenant pas d'argument et renvoyant l'indice du premier terme de la suite (u_n) dont l'écriture décimale contient chacun des chiffres de 0 à 9. On pourra essayer d'écrire un programme cohérent exploitant la fonction chiffres de la question précédente même si on n'a pas réussi à créer une version correcte de cette dernière.
8. Écrire une fonction Python **touschiffresplusieursfois(p)** prenant en argument un entier naturel p et renvoyant l'indice (mais pas la valeur !) du premier terme de la suite (u_n) dont l'écriture décimale contient (au moins) p fois chacun des chiffres de 0 à 9. On évitera de tester cette fonction sur des valeurs trop grandes de p pour ne pas faire planter les machines. Normalement, on devrait par exemple trouver touschiffresplusieursfois(3)=186 (et pour $p = 10$, ça ne donne « que » 568, ça ne monte en fait pas très vite).

II. Exploitation d'une formule explicite.

On peut démontrer la formule suivante sur les termes de la suite de Fibonacci : $\forall n \in \mathbb{N}$, $u_n = \frac{1}{\sqrt{5}}(\varphi^n - \psi^n)$, où $\varphi = \frac{1+\sqrt{5}}{2}$ et $\psi = \frac{1-\sqrt{5}}{2}$. Comme $\psi \simeq -0.6$, on peut en déduire facilement que u_n est en fait le nombre entier le plus proche du nombre réel $\frac{1}{\sqrt{5}} \times \varphi^n$.

1. Écrire une fonction Python **plusprocheentier(x)** prenant en argument un nombre flottant x, et renvoyant l'entier le plus proche de x.
2. En déduire une fonction Python **fibobis(n)** calculant la valeur de u_n comme entier le plus proche de $\frac{\varphi^n}{\sqrt{5}}$. Pour les calculs de racine carrée, on pourra utiliser au choix la puissance $\frac{1}{2}$, ou la fonction **sqrt** importée du module **math**.
3. La fonction fibobis donne en fait des valeurs erronées de u_n pour des valeurs trop grandes de n (on ne demande pas d'expliquer pourquoi). Écrire une suite de commandes Python permettant de déterminer quel est le plus petit entier n pour lequel on obtient une mauvaise valeur (on pourra bien sûr reprendre pour cette question des programmes écrits dans la première partie du sujet).
4. La formule donnée au début de cette partie permet de prouver que $\lim\limits_{n\to+\infty} \frac{u_{n+1}}{u_n} = \varphi$. On peut par ailleurs démontrer que, en posant $v_n = \frac{u_{n+1}}{u_n}$, on aura toujours $v_{2n} > \varphi$ et $v_{2n+1} < \varphi$, de sorte que l'inégalité $|v_n - \varphi| \leqslant |v_{n+1} - v_n|$ est toujours vérifiée. Écrire une fonction Python **approxor(e)** prenant en argument un nombre flottant e et calculant une valeur approchée à e près du nombre d'or φ en utilisant son approximation par les termes de la suite (v_n). La valeur de e devra bien sûr être choisie suffisamment proche de 0 pour que le calcul ait un intérêt en pratique (pour $e = 0.001$, on devrait par exemple obtenir $\varphi \simeq 1.618$).

III. Exploitation du calcul matriciel.

En notant $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, on prouve facilement que, $\forall n \in \mathbb{N}$, $A^n = \begin{pmatrix} u_{n-1} & u_n \\ u_n & u_{n+1} \end{pmatrix}$, où u_n désigne toujours le terme d'indice n de la suite de Fibonacci. Pour les programmes demandés dans

cette partie du sujet, la matrice deux lignes deux colonnes $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ sera représentée en Python par la liste de listes [[a,b],[c,d]]. Quand l'énoncé utilisera le terme « matrice », il sera donc sous-entendu qu'il s'agit de listes de deux listes à deux élements.

1. Écrire une fonction Python **produitmat(M1,M2)** prenant en arguments deux matrices $M1$ et $M2$ et renvoyant le produit matriciel $M1M2$ (toujours sous forme de liste). On ne demande PAS un programme fonctionnant sur des matrices de taille quelconque, mais uniquement pour des matrices à deux lignes et deux colonnes.
2. Écrire une fonction Python **puissancemat(M,n)** prenant en arguments une matrice M et un entier naturel n, et renvoyant la matrice M^n. On veillera à écrire une fonction gérant correctement le cas $n = 0$.
3. En déduire une fonction Python **fibomat(n)** prenant comme argument un entier naturel n et renvoyant la valeur de u_n, en exploitant les puissaces de la matrice A définie ci-dessus.

IV. Exploitation des coefficients binômiaux.

Une identité mathématique plus inattendue affirme que $u_n = \sum_{k=0}^{p} \binom{n-1-k}{k}$, où l'indice p est la partie entière du nombre $\frac{n-1}{2}$, et la notation à l'intérieur de la somme désigne comme vous en avez l'habitude les coefficients binômiaux définis par $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

1. Écrire une fonction Python **factorielle(n)** prenant en argument un entier naturel n, et renvoyant la valeur de $n!$. Il est tout à fait autorisé (mais bien sûr pas indispensable) d'écrire une fonction récursive.
2. Écrire une fonction Python **coefbin(n,k)** prenant comme arguments deux entiers naturels n et k et renvoyant la valeur du coefficient binômial $\binom{n}{k}$. On veillera à ce que la fonction renvoie la valeur 0 si $n < k$ ou si n ou k sont des entiers strictement négatifs.
3. Déduire des questions précédentes une fonction Python **fibobino(n)** prenant en argument un entier naturel n, et renvoyant la valeur de u_n calculée à l'aide de la formule sommatoire donnée ci-dessus.
4. Le calcul de coefficients binômiaux à l'aide de factorielles est en fait particulièrement lent. Pour l'améliorer, il est plus judicieux d'exploiter la relation de Pascal $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$. Écrire une fonction Python **listebino(n)** prenant en argument un entier naturel n, et renvoyant la liste de tous les coefficients binômiaux $\binom{n}{k}$ lorsque k varie entre 0 et n (le résultat renvoyé doit donc être une liste de longueur $n+1$) calculés de proche en proche à l'aide de la relation de Pascal. L'idée est de faire faire à Python le calcul comme on l'effectue « à la main » quand on construit le triangle de Pascal : on part de la liste à un élément [1], et à chaque étape de calcul on construit une liste contenant un élément de plus que la précédente, et dont les éléments sont les sommes d'éléments contigüs de la liste précédente. Pour vérifier la correction de votre programme, on donne la liste des coefficients binômiaux $\binom{10}{k}$: [1,10,45,120,210,252,210,120,45,10,1]. Les initiatives utilisant d'autres propriétés mathématiques des coefficients binômiaux (symétrie notamment) sont autorisées pour cette question.