Feuille_Oral_Info_G2E : préparation à l'épreuve orale d'informatique

Sans ordinateur. Pour chaque exercice : 20 min de préparation, puis présentation au tableau en 10 min environ. On choisit des noms de variables explicites.

Ex 1 : Les polynômes

On représente un polynôme par la liste de ses coefficients, par degré croissant. Par exemple $3x^2+4x-7$ est codé par `L = [-7, 4, 3]`.

1) Écrire une fonction qui, à un polynôme (donné par sa liste de coefficients) et à un réel x, associe la valeur du polynôme en x.
2) Écrire une fonction qui renvoie la liste des coefficients de la somme de deux polynômes :
   a. lorsque les deux listes ont la même longueur ;
   b. lorsqu'elles ont des longueurs différentes.
3) Écrire une fonction qui renvoie la liste des couples `(coefficient, degré)` des termes de coefficient non nul. Exemple : pour $3x^{10}+2x^5+3x-7$, la fonction renvoie `[(-7,0), (3,1), (2,5), (3,10)]`.
4) Écrire une fonction qui compare deux polynômes et renvoie `True` s'ils sont égaux, `False` sinon. *(deux listes de longueurs différentes peuvent représenter le même polynôme)*
5) On pose $E_n(x)=\sum_{i=0}^{n}\frac{x^i}{i!}$ (somme partielle de la série exponentielle). Écrire une fonction `factorielle`, puis une fonction renvoyant $E_n(x)$.
6) Reprendre la question précédente en établissant une relation de récurrence sur le terme général $\frac{x^i}{i!}$, afin d'éviter de recalculer la factorielle à chaque étape.

– –

Ex 2 : Les chaînes de caractères

1) Écrire une fonction qui engendre un mot de passe de la forme `nomdusite!Xcode`, où `X` est la dernière lettre du nom du site. Exemple : pour « laposte » et le code `"1234"`, la fonction renvoie `"laposte!e1234"`.
2) Écrire une fonction qui renvoie `True` si au moins un caractère apparaît en double dans la chaîne donnée, `False` sinon.
3) a. Écrire une fonction qui renvoie `True` si la chaîne donnée est un palindrome.
   b. Écrire une fonction qui compte le nombre de palindromes dans une liste de mots.
4) Écrire une fonction qui échange la première et la seconde moitié des lettres d'une chaîne (si la longueur est impaire, le caractère central reste à sa place). Exemples : `BONJOUR` → `OURJBON` et `CACHER` → `HERCAC`.
5) Deux mots sont *anagrammes* si l'un s'obtient en permutant les lettres de l'autre.
   a. Écrire une fonction qui détermine si deux mots sont anagrammes, dans le cas où aucune lettre ne se répète.
   b. Reprendre dans le cas général (les lettres peuvent se répéter).

Ex 3 : Approximation de π

La formule de Leibniz donne $\dfrac{\pi}{4} = \sum_{i=0}^{+\infty} \dfrac{(-1)^i}{2i+1}$, c'est-à-dire $\pi = 4\sum_{i=0}^{+\infty} T_i$ avec $T_i = \dfrac{(-1)^i}{2i+1}$.

1) Écrire une fonction qui renvoie une valeur approchée de π en sommant les termes T_i pour i allant de 0 à n.
2) Établir une relation de récurrence reliant T_{i+1} à T_i, puis récrire la fonction en l'utilisant (sans recalculer $(-1)^i$ à chaque étape).
3) Modifier la fonction pour qu'elle s'arrête dès que la valeur approchée diffère de π de moins de ε, et qu'elle renvoie le couple formé de la valeur approchée et du nombre d'opérations effectuées.
4) On dispose d'une liste `dec` contenant les 10 000 premières décimales de π (un chiffre par élément).
   a. Écrire une fonction qui vérifie que la somme des 20 premières décimales vaut 100.
   b. Écrire une fonction qui renvoie le rang du premier 0 rencontré dans `dec`, ou `False` s'il n'y en a aucun.
   c. Écrire une fonction qui range ces 10 000 décimales dans un tableau (liste de listes) formé de lignes de 10 chiffres. *(on utilisera une double itération)*
   d. Une date de naissance est codée par la liste de ses chiffres (par exemple le 7 mars 2006 : `[0,7,0,3,2,0,0,6]`). Écrire une fonction qui renvoie le rang de la première apparition de cette suite de chiffres dans `dec`, ou `False` si elle n'y figure pas.

– –

Ex 4 : Le rendu de monnaie

Un système de monnaie est donné par une liste de valeurs de pièces, par exemple `pieces = [1, 2, 5, 10, 20, 50]` (en centimes).

1) Écrire une fonction qui, à une somme entière s et au système `pieces`, associe le nombre de pièces utilisées par la méthode « gloutonne » (à chaque étape, on rend la plus grande pièce possible). *(les opérateurs `//` et `%` sont utiles)*
2) Modifier la fonction pour qu'elle renvoie la liste des pièces rendues, de la plus grande à la plus petite.
3) Écrire une fonction qui vérifie qu'une liste de pièces rendues correspond bien à la somme demandée.
4) Écrire une fonction qui renvoie la valeur de la plus grande pièce inférieure ou égale à une somme donnée, ou `False` si aucune pièce ne convient.

– –

Ex 5 : Notes d'une classe

Les notes d'une classe sont rangées dans une liste, par exemple `notes = [12, 8, 15, 9, 17, 6]`.

1) Écrire une fonction qui calcule la moyenne d'une liste de notes.
2) Écrire une fonction qui renvoie la meilleure note ainsi que le rang de l'élève qui l'a obtenue, sans utiliser `max`.
3) Écrire une fonction qui renvoie la liste des rangs des élèves ayant la moyenne (note supérieure ou égale à 10).
4) Écrire une fonction qui renvoie `True` si toutes les notes sont strictement supérieures à 5, `False` sinon. *(attention au placement du `return`)*
5) On dispose maintenant des notes de plusieurs classes, rangées dans une liste de listes. Écrire une fonction qui renvoie `True` si chaque classe possède au moins un élève ayant la moyenne. *(double itération)*

– –

Ex 6 : La suite de Syracuse

Pour un entier $n \geqslant 1$, on définit le terme suivant de la suite de Syracuse par : $n/2$ si n est pair, $3n+1$ si n est impair. Une célèbre conjecture affirme que, partant de n'importe quel entier $n \geqslant 1$, la suite finit toujours par atteindre 1.

1) Écrire une fonction `etape` qui, à un entier $n \geqslant 1$, associe le terme suivant de la suite. *(les opérateurs `%` et `//` sont utiles ; le résultat doit rester un entier)*
2) Écrire une fonction qui renvoie la liste des termes de la suite partant de n, jusqu'à atteindre 1 (inclus). *(on pense à `append`)*
3) Écrire une fonction `temps_de_vol` qui renvoie le nombre d'étapes nécessaires pour atteindre 1 en partant de n.
4) Écrire une fonction `altitude_maximale` qui renvoie le plus grand terme atteint par la suite partant de n, sans utiliser `max`.
5) Écrire une fonction qui, à un entier N, associe l'entier $n \leqslant N$ dont le temps de vol est le plus long, ainsi que ce temps de vol.

– –

Ex 7 : Les chiffres d'un entier

Dans tout l'exercice, n désigne un entier naturel non nul.

1) Écrire une fonction qui renvoie le nombre de chiffres de n, sans convertir n en chaîne de caractères. *(on divise par 10 tant que c'est possible)*
2) Reprendre la question précédente en utilisant la conversion `str(n)`. *(attention : `str(n)`, et non `"n"`)*
3) Écrire une fonction `somme_chiffres` qui renvoie la somme des chiffres de n :
   a. par une méthode arithmétique (opérateurs `%` et `//`) ;
   b. en parcourant la chaîne `str(n)`. *(penser à reconvertir chaque caractère avec `int`)*
4) On remplace n par la somme de ses chiffres, et on recommence jusqu'à obtenir un entier inférieur à 10. Écrire une fonction qui renvoie le couple formé du nombre d'étapes effectuées et du chiffre final. Exemple : pour $n = 976$, on obtient $976 \to 22 \to 4$, soit 2 étapes et le chiffre 4.
5) Écrire une fonction qui renvoie le plus petit entier nécessitant au moins k étapes, ou `False` si aucun entier inférieur à 10^6 ne convient. *(attention au placement du `return`)*

– –

Ex 8 : Listes et références

1) Écrire une fonction `plafonne` qui, à une liste de nombres `valeurs` et à un nombre m, remplace dans la liste chaque élément strictement supérieur à m par m. Que doit renvoyer cette fonction ? *(une fonction qui modifie une liste en place n'a pas à la renvoyer)*
2) Écrire une fonction `echange` qui échange les éléments d'indices i et j d'une liste.
3) En déduire une fonction `miroir` qui inverse l'ordre des éléments d'une liste en place, sans créer de nouvelle liste. *(attention à ne pas échanger deux fois)*
4) Écrire une fonction qui, sans modifier la liste donnée, renvoie une *nouvelle* liste contenant les mêmes éléments en ordre inverse. Comparer avec la question précédente.
5) Prévoir, sans l'exécuter, ce qu'affiche le script suivant, et expliquer :

```
L = [1, 2, 3] ;  M = L ;  M.append(4) ;  print(L)
```

– –

Ex 9 : Simulation : lancers de dés

On dispose de la fonction `randint` du module `random` : `randint(1, 6)` renvoie un entier aléatoire entre 1 et 6, chaque valeur étant équiprobable.

1) Écrire une fonction qui simule le lancer de deux dés et renvoie la somme des résultats.
2) Écrire une fonction qui répète N fois l'expérience précédente et renvoie la fréquence d'apparition d'une somme égale à 7.
3) Écrire une fonction qui simule des lancers successifs d'un dé jusqu'à l'obtention d'un 6, et renvoie le nombre de lancers effectués.
4) En répétant N fois l'expérience précédente, écrire une fonction qui estime le nombre moyen de lancers nécessaires pour obtenir un 6. Quelle valeur théorique attend-on ?
5) Écrire une fonction qui effectue N lancers d'un dé et renvoie la liste des fréquences d'apparition de chacune des six faces. *(attention au décalage entre faces et indices)*

– –

Ex 10 : Changement de base

Soit $b \geqslant 2$ une base. Tout entier $n \geqslant 1$ s'écrit de façon unique $n = \sum_{i=0}^{p} c_i\, b^i$ avec $0 \leqslant c_i < b$ et $c_p \neq 0$: les c_i sont les chiffres de n en base b.

Convention d'ordre. L'écriture usuelle de n affiche les chiffres par poids *décroissants* : $c_p\, c_{p-1} \cdots c_1\, c_0$ (le chiffre des unités c_0 est à droite). En Python, on codera au contraire cette écriture par la liste `[c0, c1, ..., cp]`, *poids croissants* : le chiffre des unités c_0 est en tête de liste, le chiffre de plus grand poids c_p en dernière position. Ainsi $13 =$ `1101` en base 2, codé par la liste `[1, 0, 1, 1]`.

En base 16 (hexadécimal), les chiffres de valeur 10 à 15 sont notés `A`, `B`, `C`, `D`, `E`, `F`.

1) Écrire une fonction qui renvoie la liste des chiffres de n en base 2, obtenus par divisions successives. On remarquera que les divisions produisent naturellement les chiffres par poids croissants, c'est-à-dire dans l'ordre de la liste. *(les opérateurs `%` et `//` font tout le travail; on pense à `append`)* Exemple : pour $n = 13$, la fonction renvoie `[1, 0, 1, 1]`.
2) Généraliser : écrire une fonction qui renvoie la liste des chiffres de n dans une base b quelconque.
3) Réciproquement, écrire une fonction qui, à une liste de chiffres `c = [c0, c1, ..., cp]` (poids croissants, conformément à la convention) et à la base b, associe l'entier n correspondant.
4) *Méthode de Horner.* On remarque que, par exemple, $c_0 + c_1 b + c_2 b^2 + c_3 b^3 = c_0 + b\,\big(c_1 + b\,(c_2 + b\,c_3)\big)$. Réécrire la fonction précédente en parcourant la liste *de la dernière case vers la première* (donc des poids forts vers les poids faibles), sans calculer aucune puissance. Combien de multiplications chaque version effectue-t-elle ? *(c'est le même calcul que la valeur d'un polynôme en $x = b$)*
5) On donne la chaîne `chiffres = "0123456789ABCDEF"`.
   a. Écrire une fonction qui, à un caractère hexadécimal `c`, associe sa valeur entière (par exemple `"B"` $\to$ 11), ou `None` si le caractère est invalide.
   b. En déduire, par la méthode de Horner, une fonction qui convertit en entier une chaîne hexadécimale écrite dans l'ordre usuel (poids forts à gauche). On remarquera qu'ici, contrairement à la question 4, la lecture de gauche à droite parcourt déjà les chiffres des poids forts vers les poids faibles. Exemple : `"1A3"` $\to$ 419.
6) Écrire une fonction qui, à un entier $n \geqslant 1$, associe la chaîne de son écriture hexadécimale dans l'ordre usuel (poids forts à gauche). Exemple : 419 $\to$ `"1A3"`. *(les divisions successives produisent les chiffres des unités d'abord : on construira donc la chaîne en ajoutant chaque nouveau caractère* devant *les précédents)*

– –

Ex 11 : Les polynômes sont représentés simplement par des listes de nombres.

Par exemple :

- $3X^2 + 2X - 5$ est représenté par `[-5, 2, 3]` mais aussi par `[-5, 2, 3, 0, 0]` ou ...
- Le polynôme nul est représenté par `[0]` mais aussi par `[]` ou `[0, 0]` ...

La donnée de cette liste définie entièrement le polynôme associé.

1) Ecrire une fonction Python `deg(P)` qui prend pour argument une liste `P` représentant un polynôme et qui renvoie son degré.
2) Ecrire une fonction Python `P_de_z(P, z)` qui prend pour argument une liste `P` représentant un polynôme et un nombre `z` qui renvoie la valeur de $P(z)$.
3) *Méthode de Horner.* On remarque que, par exemple, $c_0 + c_1X + c_2X^2 + c_3X^3 = c_0 + X\left(c_1 + X\left(c_2 + X\,c_3\right)\right)$. Réécrire la fonction précédente en parcourant la liste *de la dernière valeur vers la première*, sans calculer aucune puissance. Combien de multiplications chaque version effectue-t-elle ?
4) Ecrire une fonction Python `somme(P, Q)` qui prend pour arguments deux listes représentant deux polynômes P et Q et qui renvoie une liste représentant le polynôme $P + Q$.
5) Ecrire une fonction Python `produit(P, Q)` qui prend pour arguments deux listes représentant deux polynômes P et Q et qui renvoie une liste représentant le polynôme $P \times Q$.
6) Ecrire une fonction Python `puissance(P, n)` qui prend pour arguments une liste représentant un polynômes P et un entier n qui renvoie une liste représentant le polynôme P^n.
7) Ecrire une fonction Python `derive(P)` qui prend pour arguments une liste représentant un polynômes P qui renvoie une liste représentant le polynôme P'.

Ex 12 : On définit la fonction S sur les entiers naturels : $S(p)$ est la somme des carrés des chiffres de p. *Exemple :* $S(450) = 4^2 + 5^2 + 0^2 = 41$.

1) Écrire une fonction `S(p)` prenant en argument un entier p et renvoyant la somme des carrés de ses chiffres.
2) On définit la suite $t_0 = n$, $t_{k+1} = S(t_k)$. On admet que cette suite finit toujours par atteindre 1 (auquel cas n est dit heureux) ou par entrer dans un cycle contenant 89 (auquel cas n n'est pas heureux).
   Écrire une fonction `est_heureux(n)` renvoyant `True` si n est heureux et `False` sinon.
3) Écrire une fonction `nb_heureux(n)` prenant en argument un entier n et renvoyant le nombre d'entiers heureux dans $[\![1, n]\!]$.
4) *(Discussion.)* Combien d'entiers heureux y a-t-il parmi $[\![1, 100]\!]$? Proposer une amélioration de `est_heureux` pour éviter de recalculer des valeurs déjà connues (en utilisant une liste de valeurs déjà testées).

Ex 13 : On considère une liste L dont les valeurs sont prises dans l'intervalle d'entiers $[\![0; k-1]\!]$.

1) Écrire une fonction `occurrence(L, k)` qui prend en entrée une liste `L` et un entier `k` et qui renvoie la liste des nombres d'occurrences des entiers de `0` à `k-1` dans la liste `L`

On supposera que `L` est une liste de nombres et que `0 <= k < len(L)`, la fonction n'aura pas à le vérifier.

2) Si la fonction précédente n'a pas été faite avec une seule boucle, la refaire avec une seule boucle, sinon passer à la question suivante.
3) Écrire en utilisant la fonction `occurrence(L, k)` une fonction `tri_denombre(L)` qui trie dans l'ordre croissant la liste `L` passée en argument.

Ex 14 : 1) Écrire une fonction `insere(L, j)` qui prend en entrée une liste `L` et un entier `j` et qui insère `L[j]` dans la sous-liste `L[0:j]` déjà triée de sorte que `L[0:j+1]` soit triée.

On supposera que `L` est une liste de nombres, que `j<len(L)` et que `L[0:j]` est triée, la fonction n'aura pas à le vérifier.

2) Si la fonction précédente n'a pas été faite avec uniquement la liste `L` *(sans liste auxiliaire)*, la refaire avec uniquement la liste `L`, sinon passer à la question suivante.

3) Écrire en utilisant la fonction `insere` une fonction `tri_insertion(L)` qui trie dans l'ordre croissant la liste `L` passée en argument.

Sans préparation.

Ex 1 : Les vecteurs de $\mathbb{R}^n$ sont représentés ici par des listes de flottants de longueur n.

1) Ecrire une fonction Python `colineaire_2(u, v)` qui prend en entrée deux vecteurs de $\mathbb{R}^2$ et qui renvoie `True` si u et v sont colinéaires et `False` sinon.
2) Même question pour $n = 3$.
3) Même question pour un n quelconque.

Ex 2 : Etant donné `L` une liste de listes de nombres.

1) Ecrire une fonction Python `Contient_negatif(L)` qui renvoie `True` si toutes les sous-listes contiennent au moins un nombre strictement négatif et `False` sinon.
2) Ecrire une fonction Python `Somme_totale(L)` qui renvoie la somme de tous les éléments de toutes les sous-listes.
3) Ecrire une fonction Python `Liste_max(L)` qui renvoie la sous-liste dont la somme est maximale.

Ex 3 : Écrire une fonction `gene(n)` qui renvoie une chaîne de n caractères formée aléatoirement et équiprobablement des caractères `'A'`, `'C'`, `'G'`, `'T'`.

Puis écrire une fonction `nbAC(g)` qui prend en entrée une chaîne formée de ces quatre caractères et renvoie le nombre d'occurrences de la sous-chaîne `'AC'`.

Exemple : `nbAC('GAGCACCCTACTTGGCGCGA')` doit renvoyer `2`.

Ex 4 : On dispose d'une liste `L` dont les éléments sont des entiers compris entre 0 et k (donné en paramètre). **L'utilisation de `max` et `count` est interdite.**

Écrire une fonction `plus_frequent(L, k)` qui renvoie l'élément le plus fréquent de `L` (ou l'un d'eux en cas d'égalité).

(Construire une liste de compteurs `freq` de longueur $k+1$ avec une boucle.)

Exemple : `plus_frequent([0, 4, 0, 1, 4], 4)` peut renvoyer `0` ou `4`.

Ex 5 : Écrire une fonction `somme_cumul(L)` qui prend en entrée une liste d'entiers L et renvoie une liste M de même longueur telle que

$$M[i] = \sum_{k=0}^{i} L[k] \quad \text{pour tout indice } i.$$

Exemple : `somme_cumul([3, 1, 4, 1, 5])` doit renvoyer `[3, 4, 8, 9, 14]`.

Ex 6 : Soit `L` une liste de nombres, de longueur n.

On dit qu'un entier p (avec $1 \leqslant p \leqslant n$) est une *période* de `L` lorsque `L[i]` et `L[i+p]` sont égaux pour tout indice i vérifiant $i + p < n$. Par exemple, `[2, 5, 2, 5, 2]` admet 2 pour période, et `[1, 1, 1]` admet 1 pour période.

1) Ecrire une fonction Python `est_periode(L, p)` qui renvoie `True` si p est une période de `L`, et `False` sinon.
2) En déduire une fonction Python `plus_petite_periode(L)` qui renvoie la plus petite période de `L`. On remarquera que n est toujours une période de `L` : la fonction renvoie donc toujours un résultat.

Ex 7 : Une matrice est codée par une liste de listes de nombres.

1) Ecrire une fonction Python `test_sym(M)` qui renvoie `True` si `M` est symétrique et `False` Sinon.
2) On dit qu'une matrice carrée réelle $M = (m_{i,j})$ est *diagonale à dominante stricte par lignes* si :

$$\text{Pour tout indice } i, \quad |m_{i,i}| > \sum_{j \neq i} |m_{i,j}|$$

Écrire une fonction Python `diago_dominante_stricte(M)` qui renvoie `True` si une matrice M vérifie la condition de dominance stricte par lignes et `False` sinon.