Devoir surveillé 1

Durée : 1H. Calculatrices interdites. **Les candidats sont invités à porter une attention particulière à la rédaction : les copies illisibles ou mal présentées seront pénalisées.**

Lisez l'énoncé **attentivement** avant de répondre à chaque question.

De nombreuses applications informatiques requièrent l'évaluation en de nombreux points de polynômes. Le but de ce devoir est de donner des algorithmes efficaces permettant de ce faire.

I Etude préliminaire

Dans tout ce devoir, on considère comme opération élémentaire : la somme, le produit, la division (entière ou flottante, ou le calcul du reste dans la division entière), la différence, l'affectation d'une variable.

Exercice 1
On considère la fonction suivante :

```
def expo(x, n):
    res = 1
    j = n
    while j > 0:
        res = res * x
        j = j - 1
    return res
```

1. A l'aide d'un tableau dont les colonnes seront `res` et `j`, expliciter les valeurs prises par ces variables lors du calcul de `expo(2, 3)`.
2. On note $j_1, j_2, \dots$ les valeurs de j à la fin de $1, 2, \dots$ passages dans la boucle. De même pour res. Montrer que pour tout $k \leqslant n$ on a $x^{j_k} \times res_k = x^n$ à la fin du $k$ième passage dans la boucle.
3. On note $C(n)$ le nombre d'opérations élémentaires (voir le préambule pour la définition) nécessaires au calcul de x^n par cette fonction. Montrer que $C(n) = 4n + 2$.

Exercice 2
Dans la suite du devoir nous allons manipuler des listes python. En guise d'échauffement, nous allons traiter quelques questions de cours.

1. Écrire une fonction **maxi(L)** qui prend comme argument une liste `L` de nombres et retourne la valeur maximale stockée dans `L`
2. On considère encore que `L` est une liste de nombres.

```
def mystere(L):
    s = 0
    for i in range(len(L)):
        s = s + L[i]
    return s
```

   (a) Quel est le résultat de l'appel de `mystere` sur la liste `[4, -1, 2, 3]` ?
   (b) Expliquer succinctement quelle est la valeur de retour de la fonction `mystere`.
   (c) Écrire une autre version de cette fonction qui n'utilise pas la fonction `range`.
3. Donner une suite d'instructions python permettant de construire la liste contenant, dans l'ordre, les 1000 premiers entiers naturels au carré : $[0, 1, 4, \dots, 1000^2]$

II Polynômes

Représentation des polynômes

On souhaite manipuler des fonctions polynomiales en python. Le premier pas est de s'accorder sur la manière de les représenter en mémoire.
La fonction à représenter est $f : x \mapsto a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$.
On la représente en python par la liste suivante de longueur $n + 1$: $[a_0, a_1, \dots, a_n]$.
Ainsi la fonction $f : x \mapsto -1 + x^2$ est représentée par la liste $[-1, 0, 1]$.

Exercice 3
1. Quel fonction polynomiale, notée g, est représentée par `L = [0,1,2,3,4]` ?
2. Pour cette fonction g, que vaut $g(1)$? Comment obtenir ce résultat par un appel de fonction (déjà croisée) sur la liste `L` ?

III Evaluation des polynômes

Approche naïve

Il s'agit ici d'implémenter la solution la plus simple à notre problème.

Exercice 4
1. Pour une fonction polynomiale f et un nombre x, on veut calculer $f(x) = a_0 + a_1 x + \dots + a_n x^n$. Pour ce faire on va créer une fonction `evalue` qui prend `L` (la liste représentant la fonction f) et `x` comme arguments et retourne $f(x)$.
   Décrire les variables nécessaires à ce calcul en précisant leurs rôles lors de l'exécution.
2. Écrire la fonction `evalue(L, x)` en utilisant `expo` pour calculer les puissances de x.

3. On note $\tau_1(n)$ le nombre d'opérations élémentaires nécessaires à l'évaluation de $f(x)$ quand la liste représentant f est de longueur n (donc le degré de f est $n-1$)). Exprimer $\tau_1(n)$ en fonction de n et des $\mathrm{C}(i)$ pour $i \in [\![0, n]\!]$ (voir la définition de C dans l'exercice 1).
4. Calculer explicitement $\tau_1(n)$ en fonction de n.
5. Si on utilise plutôt `x**i` pour calculer x^i, on obtient un nombre d'opérations élémentaires $\tau_2(n) \approx 5n\log_2(n)$.

   En prenant comme approximation $2^{10} \approx 1000$, calculer le rapport $\frac{\tau_1(1000)}{\tau 2(1000)}$.

Approche plus raffinée

Exercice 5
On considère la fonction python

```
def evalue_rapide(L, x):
    p = 1
    for i in range(len(L)):
        res = res + L[i] * p
        p = p * x
    return res
```

1. Si on exécute maintenant `evalue_rapide([0,1,2,3,4], 1)` dans la console, nous obtenons une erreur : `NameError`. Expliquer pourquoi et rectifier le code précédent pour que la valeur de retour soit bien $f(x)$ (avec la même convention de notation que dans l'exercice précédent).
2. Calculer $\tau_3(n)$, le nombre d'opérations élémentaires nécessaires au calcul de $f(x)$ quand la liste représentant f est de longueur n. On pourra utiliser au choix la version correcte ou la version de l'énoncé pour calculer τ_3.

   Calculer, dans la même approximation qu'à l'exercice précédent, $\frac{\tau_2(1000)}{\tau_3(1000)}$.

Algorithme de Horner

Une manière encore plus raffinée de résoudre ce problème est de changer l'écriture de la fonction polynomiale f. On remarque que

$$f(x) = a_0 + x \times (a_1 + x \times (a_2 + \dots))$$

Exercice 6
Voyons ceci de plus près.

1. Ecrire de cette manière l'expression $x^3 + 2x^2 + x - 1$.
2. Une implémentation python de cette écriture peut être

```
def horner(L, x):
    res = 0
    for i in range(len(L)-1, -1, -1):
        res = x*res + L[i]
    return res
```

Cette utilisation du range est un peu particulière : i prend toutes les valeurs entières de $len(\mathrm{L}) - 1$ jusqu'à 0 en décroissant.
On prend $\mathrm{L} = [0, 1, 2, 3]$ et $x = 2$. Donner un tableau représentant l'évolution des valeurs des variables `i` et `res`.

3. Donner la complexité (ie le nombre d'opérations élémentaires) de cette fonction en fonction de la longueur de L notée n. On la notera $\tau_4(n)$.
   Avons-nous amélioré la situation ?