Quelques programmes classiques

Vous trouverez ici quelques exercices d'entrainement : tous ces thèmes ont été abordés dans l'année et les programmes demandés sont assez classiques. Essayez de vous appliquer sur la syntaxe et il est toujours bien vu, même si vous allez vite, de renseigner un peu la fonction :

1. on fait attention à renseigner la classe des arguments à la définition d'une fonction, par exemple : `def puiss(A:array,n:int):`,
2. on peut ajouter une ou deux pré-conditions : `assert n>=0 and isinstance(n,int)`
3. sans oublier de commenter un ou deux passages... pas plus, car on perd alors le fil.

Extrait du rapport Mines-Ponts - 2021

- Si certaines copies sont très faibles (voire presque vides), certaines sont excellentes et frisent parfois la perfection. La longueur et la difficulté du sujet étaient ainsi tout à fait adaptées à ce type d'épreuve, ce qui a permis de classer les candidats.
- Cette année encore, le jury souhaite souligner l'importance de la présentation des copies. Certaines sont très brouillonnes, sales, voire parfois illisibles. Un nombre trop important de ratures nuit forcément à la lecture des codes Python produits, et peut même provoquer des erreurs de syntaxe. On peut certes tolérer quelques ratures propres (correction d'un oubli, d'une erreur de syntaxe), qui ne nuisent pas à la poursuite de la lecture ni à la structure des codes proposés, mais un code trop difficile à déchiffrer (excès de ratures ou de rajouts par le biais de flèches ou d'astérisques) est forcément sanctionné.
- De la même façon, une erreur ponctuelle de syntaxe (oubli d'une parenthèse fermante) peut être tolérée. En revanche, l'absence récurrente des parenthèses (en écrivant par exemple systématiquement `for i in range n` ou `len L`) a été sanctionnée.
- L'importation des bibliothèques en Python doit être maîtrisée. Il existe plusieurs façons de procéder, qui ne doivent pas être confondues. On pourra par exemple écrire `import math`, `import math as m`, `from math import *` ou encore `from math import cos,sin`. Il est cependant demandé que la syntaxe de l'appel aux fonctions Python de ces bibliothèques soit en adéquation avec la syntaxe d'importation choisie.
- Le sujet demandait explicitement de manipuler des listes. À ce titre, il n'était pas opportun d'utiliser le module `numpy`. Les candidats doivent maîtriser la manipulation des listes, notamment :
  - la construction d'une liste élément par élément.Par exemple, l'initialisation d'une liste `L = [ ]` suivie, dans une boucle `for`, d'une affectation `L[i] = elt` provoque une erreur ;
  - l'ajout d'un élément à la fin d'une liste. Comme indiqué dans les rapports des années précédentes, la syntaxe `L.append(elt)` est à privilégier. D'une part, elle est plus efficace, mais elle est également moins source d'erreurs. L'emploi de la syntaxe `L = L + [elt]`(ou `L+ = [elt]`) a par exemple provoqué beaucoup d'oubli de crochets, quand `elt` était lui-même une liste :
  - les erreurs de syntaxe "à la `numpy`" : l'addition terme à terme de deux listes de même taille (respectivement la multiplication terme à terme d'une liste par un flottant ou un entier) ne peut pas se faire à l'aide d'une syntaxe du type `L1 + L2` (respectivement `a * L`), réservée aux tableaux `numpy`. L'accès à un élément d'une liste de listes se fait quant à lui *via* une syntaxe du type `L[i][j]`, et pas `L[i, j]` ;
  - le caractère modifiable des listes ; il s'agit d'un aspect subtil de Python, dont les candidats doivent avoir pris conscience au cours de leur formation. Par exemple, quand l'énoncé demande de créer une nouvelle liste, on ne peut pas juste modifier la liste entrée en paramètre d'une fonction ; de même, l'affectation `L1 = L2` ne permet pas de créer une copie indépendante de la liste `L2` ;
  - la modification des éléments d'une liste ; la commande `for elt in L : elt = float(elt)` n'apporte par exemple aucun changement à la liste `L`.
- Le jury a remarqué cette année un nombre croissant de copies dans lesquelles les affectations des variables sont faites à l'envers ; par exemple, si `L` était une liste placée en paramètre d'une fonction, beaucoup de candidats ont écrit `L = a, b, c, d` au lieu de `a, b, c, d = L` pour affecter les valeurs des éléments de `L` dans les variables `a, b, c, d`. En outre, rappelons que le choix du nom d'une variable doit être valide et pertinent. Un nom de variable ne peut être composé que de caractères alphanumériques et du caractère _ (pas de ', -, ., qui provoquent une erreur de syntaxe, ni de lettres grecques, comme φ). Enfin, rappelons que l'affectation d'une variable se fait en Python à l'aide d'un =, que beaucoup de candidats ont utilisé à tort à la place d'un == pour tester l'égalité des valeurs de deux objets.

Exercice 1 (résolution approchée par dichotomie). [ ]

Si on considère l'équation $x^2-2=0$ sur l'intervalle $[0,10]$, alors on peut obtenir une valeur approchée de $\alpha=\sqrt{2}$ par **dichotomie**.

Pour cela, on construit deux suites $(a_n)\in I^{\mathbb{N}}$ et $(b_n)\in I^{\mathbb{N}}$ telles que : a_0, b_0 désignent des valeurs grossières encadrant α et pour tout $n\in\mathbb{N}$,

$$a_{n+1}=\begin{cases} a_n, \text{ si } f(a_n)f(\dfrac{a_n+b_n}{2})\leq 0 \\ \dfrac{a_n+b_n}{2}, \text{ sinon}\end{cases} \quad \text{et } b_{n+1}=\begin{cases} \dfrac{a_n+b_n}{2}, \text{ si } f(a_n)f(\dfrac{a_n+b_n}{2})\leq 0 \\ b_n, \text{ sinon}\end{cases}$$

c'est à dire qu'à chaque étape, on doit évaluer le produit $f(a_n)f(\dfrac{a_n+b_n}{2})$ pour situer α dans l'intervalle $[a_n,b_n]$.

1. Justifier rapidement que les suites (a_n) et (b_n) sont adjacentes de limite commune $\ell=\alpha$.
2. Posons $f: x\in[0,+\infty[\longmapsto x^2-2$ et $(a_0,b_0)=(0,10)$.
   (a) Dans le langage Python, construire la fonction $dichotomie(n: int)\to float, float$ qui pour tout entier n donné, renvoie les valeurs des bornes a_n et b_n.
   (b) En déduire le programme $approx1(eps: float)\to int$ qui pour toute précision ϵ donnée, affiche les valeurs a_n et b_n tant que $|b_n-a_n|>\epsilon$, puis renvoie le plus petit entier n_0 pour lequel a_{n_0}, b_{n_0} désignent des approximations de ℓ à ϵ près.

Exercice 2 (retour sur les tris classiques). [ ]

Parmi les tris classiques, on peut aussi considérer :

- le **tri à bulles** : pour une liste donnée de nombres réels, on parcourt le tableau plusieurs fois et à chaque étape, on échange les éléments adjacents afin de les remettre dans l'ordre. Ainsi, comme des bulles, et à chaque passage, les plus "grands" éléments se placent à la fin de la liste... jusqu'à ce que la liste soit enfin triée.

  Par exemple, si $L=[10,5,8,1]$, alors le programme modifie la liste de sorte que :

  $$L\to[5,8,1,10]\to[5,1,8,10]\to[1,5,8,10]$$

- le **tri par insertion** : pour une liste donnée de nombres réels, on prend une valeur à chaque passage puis on l'insère à la bonne place dans une nouvelle liste. D'ailleurs, ce programme repose sur une fonction secondaire : **l'insertion**.

  Par exemple, si $L=[10,5,8,1]$, alors le programme modifie la liste de sorte que :

  $$10\to[10], 5\to[5,10], 8\to[5,8,10], 1\longrightarrow[1,5,8,10]$$

1. (a) Expliquer à quoi correspond cette instruction : `a,b=b,a`.
   (b) Dans le langage Python, construire le programme $tribulles(L: list)\longrightarrow list$ qui pour toute liste L donnée, renvoie une liste triée contenant les valeurs de L.
2. (a) Dans le langage Python, construire la fonction $insertion(L: list, x: float)\longrightarrow list$ qui pour tout couple (L,x) donné, insère l'élément x dans la liste **déjà triée** L.
   (b) En déduire le programme $triinsertion(L: list)\longrightarrow list$ qui pour toute liste L donnée, renvoie une liste triée contenant les valeurs de L.
3. Pour finir, on rappelle qu'il existe des tris plus efficaces, c'est par exemple le cas du **tri rapide ou quick sort**...
   Pour cela, considérons une liste L de n nombres réels. L'idée de ce tri est de choisir un élément pivot, par exemple $L[m]$ avec $m=n//2$, de la liste initiale, de l'enlever, puis de constituer deux sous-listes :
   - $L1$ constituée des éléments de L inférieurs ou égaux à $L[m]$
   - $L2$ constituée des éléments de L strictement plus grands que $L[m]$

   On trie alors récursivement chacune des sous-listes et on rassemble le tout.

   Le principe de cet algorithme, c'est qu'il repose sur le principe de **diviser pour régner** : on divise la tâche en appelant notre algorithme sur des sous-listes de ses données.

   Attention, comme il s'agit d'un algorithme récursif, **il faudra traiter la condition d'arrêt de ces appels récursifs**, c'est à dire lorsqu'une telle liste n'a pas d'élément ou n'est constituée que d'un seul élément.

   (a) Dans le langage Python, construire le programme $trirapide: (L: list)\longrightarrow list$ qui renvoie la liste des éléments de L triée par ordre croissant.
   (b) Justifier rapidement que le nombre d'appels récursifs est nécessairement fini, ce qui assurera la terminaison de notre programme.

Exercice 3 (racine d'un polynôme et valuation associée). [ ]
On considère P un polynôme non nul de $\mathbb{K}[X]$ tel que $P(X) = \sum_{k=0}^{n} a_k X^k,\ a_n \neq 0$. On rappelle que $\alpha \in \mathbb{K}$ désigne une racine de P si $P(\alpha) = 0$, c'est à dire si :

$$\sum_{k=0}^{n} a_k \alpha^k = 0$$

Plus précisément, on a même montré que α est une **racine multiple d'ordre de multiplicité** k si elle vérifie :

$$\begin{cases} P(\alpha) = 0,\ P'(\alpha) = 0,\ P^{(2)}(\alpha) = 0,\ \ldots, P^{(k-1)}(\alpha) = 0 \\ P^{(k)}(a) \neq 0 \end{cases}$$

Dans le langage Python, un polynôme peut alors être représenté par la liste de ses coefficients de sorte que :

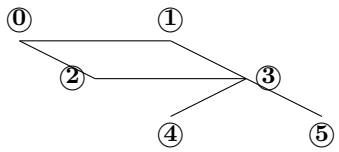
$$P = [a_0, \ldots, a_n]$$

et ainsi, $P(\alpha) = \sum_{k=0}^{n} P[k] * \alpha^k$.

1. Construire le programme $evaluation(P : list, a : float) \longrightarrow float$ qui, pour tout couple (P, a) donné, renvoie l'image de a par P.
2. Construire le programme $derive(P : list) \longrightarrow list$ qui, pour tout polynôme P donné sous la forme d'une liste, renvoie la liste des coefficients du polynôme dérivé P'.
3. En déduire la fonction booléenne $multiple(P : list, a : float) \longrightarrow bool$ qui, pour tout couple (P, a) donné, teste si a est racine multiple de P.
4. Modifier le programme précédent pour que celui-ci renvoie `True` ou `False`, ainsi que la **valuation** de la racine donnée, c'est à dire son ordre de multiplicité dans le polynôme P.

Exercice 4 (première utilisation de la matrice d'adjacence). [ ]
On considère un graphe $G = (S, A)$ à n sommets, qu'on suppose non orienté, non valué et sans boucle, et on note M_G sa matrice d'adjacence.

1. Donner, pour le graphe suivant, sa matrice d'adjacence :



2. Dans le langage Python, construire la fonction `voisins(M:array,i:int)->list` qui pour tout graphe de matrice d'adjacence M renvoie la liste des voisins du sommet i. On pourra ajouter une pré-condition pour vérifier si i est bien un sommet possible.
3. Construire les fonctions `aretes(M:array)->int` et `degretotal(M:array)->int` qui pour tout graphe de matrice d'adjacence M renvoie le nombre d'arêtes, et la somme des degrés du graphe.

Exercice 5 (approximation uniforme par les polynômes de Bernstein). [ ]
On rappelle le **théorème de Stone-Weierstrass** : pour toute fonction continue sur un segment $[a, b]$, il existe une suite de polynômes $(P_n) \in \mathbb{R}[X]^{\mathbb{N}}$ telle que :

$$P_n \overset{\|.\|_\infty}{\longrightarrow} f \text{ , c'est à dire telle que } \|P_n - f\|_\infty \longrightarrow 0$$

On note ici $f : x \in [0, 1] \longmapsto \ln(1 + x)$ et on se propose d'illustrer cette convergence uniforme.

1. Soit $n \in \mathbb{N}$. On définit la famille des **polynômes de Bernstein** par :

$$\forall k \in [\![0, n]\!],\ B_{n,k}(X) = \binom{n}{k} X^k (1 - X)^{n-k}$$

   (a) Dans le langage Python, construire la fonction $binom(k : int, n : int) \to int$ qui renvoie le coefficient binomial $\binom{n}{k}$, puis en déduire la fonction $bernstein(n : int, k : int, x : float) \to float$ qui renvoie $B_{n,k}(x)$, l'image de x par $B_{n,k}(X)$.

   (b) Construire le programme $representation(n : int) \to None$ qui affiche sur un même graphe les courbes représentatives des fonctions polynômes $(B_{n,k})_{k \in [\![0,n]\!]}$.

2. On définit alors la suite (P_n) définie par : $P_n(X) = \sum_{k=0}^{n} f(\frac{k}{n}) B_{n,k}(X)$.

   (a) Définir la fonction $polynome(n : int, x : float) \to float$ qui renvoie $P_n(x)$, l'image de x par $P_n(X)$.

   (b) En déduire le programme $approximation(n : int) \to None$ affiche sur un même graphe les courbes associées à la fonction f, ainsi que les fonctions P_n, P_{5n} et P_{10n}.