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 (revanche sur le groupe des permutations !). [ ]

Pour $n \in \mathbb{N}^*$, on note S_n le groupe des permutations de l'ensemble $[\![0, n-1]\!]$. Une permutation de S_n sera représentée en Python par une liste, dont l'élément d'indice i est l'image de i par cette permutation.

Par exemple, si on note $\sigma \in S_4$ définie par :

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 1 & 0 & 2 \end{pmatrix}$$

alors on a $\sigma(0) = 3$, $\sigma(1) = 1$, $\sigma(2) = 0$ et $\sigma(3) = 2$. Ainsi, en Python, cette permutation sera décrite par la liste des images obtenues par σ : $[3, 1, 0, 2]$.

Dans tout l'exercice, on pourra utiliser librement les tests Python du type `x in L` (respectivement `x not in L`) permettant de vérifier si x est présent dans la liste L (respectivement de vérifier si x n'est pas présent dans la liste L).

1. Si s est une liste Python représentant une permutation de S_4, quelle instruction Python permet de trouver l'image de 1 par cette permutation ? Quelle liste Python représente la transposition $(2\ 3) \in S_4$?
2. Écrire une fonction Python $composition(s1 : list, s2 : list) \longrightarrow list$prenant en entrée deux listes représentant des permutations σ_1 et σ_2 du même groupe de permutations et renvoyant la liste représentant la permutation $\sigma_1 \circ \sigma_2$. *On fera attention à longueur des permutations données en entrée.*
3. Écrire une fonction Python $inv(s : list) \longrightarrow list$ prenant en entrée une liste représentant une permutation σ et renvoyant la liste représentant σ^{-1}.
4. On souhaite tester si un sous-ensemble G de S_n est ou non un sous-groupe de S_n.
   Écrire une fonction Python $groupe(G : list) \longrightarrow bool$ prenant en entrée une liste de listes, où chaque sous-liste représente une permutation de S_n et renvoyant `True` s'il s'agit bien d'un sous-groupe de S_n, `False` sinon.
   En fait, on testera si toutes les compositions possibles restent dans G...
5. Écrire une fonction Python $cyclique(s : list) \longrightarrow list$ prenant en entrée une liste s représentant une permutation σ de S_n et construit le sous-groupe de S_n engendré par σ, c'est à dire l'ensemble des listes s^k tant que celui-ci n'est pas dans la liste.
   En effet, le groupe étant cyclique, à un moment, on retombe sur la permutation identité !

Exercice 2 (calcul des puissances d'une matrice à l'aide de la formule d'exponentiation rapide). [ ]

On rappelle qu'on peut définir l'algorithme d'exponentiation rapide par :

$$\forall n \in \mathbb{N}^*,\ a^n = \begin{cases} a^{n/2} \times a^{n/2} \text{ si } n \text{ est pair} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a \text{ si } n \text{ est impair} \end{cases}$$

et on veut adapter ce principe au calcul des puissances itérées d'une matrice A donnée. On pensera évidemment à importer le module **numpy** et on suppose dans tout l'exercice qu'on ne travaillera qu'avec des matrices carrées.

1. Dans le langage Python, construire la fonction $produit(A : array, B : array) \longrightarrow array$ qui renvoie le produit $A \times B$, résultant du produit matriciel. Attention, on ne pourra pas utiliser `dot`... mais on le programmera à la main.
   Pour rappels :
   - `(n,p)=shape(A)` renvoie le nombre de lignes et colonnes de A
   - `zeros((n,n))` renvoie la matrice nulle et `eye(n)` renvoie la matrice identité I_n
2. Construire la fonction $puiss(A : array, n : int) \longrightarrow array$ qui calcule A^n de façon itérative.
3. En utilisant l'algorithme d'exponentiation rapide, construire la fonction $puissrapide(A : array, n : int) \longrightarrow array$ qui calcule A^n de façon récursive.

Exercice 3 (décomposition primaire d'un entier). [ ]

On souhaite construire un programme *decomposition* qui renvoie la décomposition primaire d'un entier sous la forme d'une liste. Concrètement, le théorème fondamental de l'arithmétique nous donne par exemple : $20 = 2^2 \times 5$ et ainsi, l'appel de notre programme doit renvoyer :

```
In : decomposition(20)
                                    [2,2,5]
                                                              python
```

On rappelle qu'un nombre est dit **premier** si le nombre de diviseurs positifs est excatement égal à 2. En particulier, 1 n'est pas premier.

1. Construire la fonction $diviseurs(n : int) \longrightarrow list$ qui pour un entier n donné, renvoie la liste des diviseurs de n.
2. Construire la fonction $nextprime(n : int) \longrightarrow int$ qui pour un entier n donné, renvoie le premier nombre premier strictement supérieur à n.

3. Construire alors la fonction $decompositio(n : int) \longrightarrow list$ qui pour un entier n donné, renvoie la décomposition de n sous la forme d'une liste de ses facteurs premiers. On veillera :
   - à tester tous les nombres premiers p inférieurs ou égaux à n,
   - et pour chaque nombre premier p testé, on le stockera dans la liste finale, tant que celui-ci divisera n, n/p, n/p^2...
4. Pour finir, on appelle **valuation** d'un nombre premier p dans n la puissance avec laquelle intervient ce nombre dans la décompostion de n. Par exemple :

```
In : valuation(2,20); valuation(5,20); valuation(7,20)
                                2,1,0
                                                          python
```

   Construire alors la fonction $valuation(p : int, n : int) \longrightarrow int$ qui pour tout couple d'entiers donnés (p, n) renvoie la valuation de p dans n.

Exercice 4 (méthode numérique d'intégration). [ ]

On rappelle qu'il existe des **méthodes numériques d'intégration** : celles-ci consistent à prendre une subdivision (x_i) à pas constant du segment $[a, b]$, et d'approcher l'aire sous la courbe sur chacun des intervalles de la subdivision. Ainsi, si on note f une fonction de classe C^1 sur $[a, b]$, on a par exemple par le théorème de convergence des sommes de Riemann :

$$S_n = \sum_{i=0}^{n-1} \frac{(b-a)}{n} f(\frac{x_i + x_{i+1}}{2}) \underset{n\to+\infty}{\longrightarrow} \int_a^b f(t)\, dt$$

C'est la **méthode du point milieu**.

1. Dans le langage Python, construire la fonction $pointmilieu(a : float, b : float, f : function, n : int) \longrightarrow float$ qui pour tout quadruplet (a, b, f, n) donné, renvoie la valeur de S_n. On a par exemple :

```
In : pointmilieu(0,1,lambda t:sqrt(1-t**2),100)
                          0.7854842144750019
                                                          python
```

De la même façon, on cherche à construire une autre méthode d'approximation qui repose sur la loi faible des grands nombres. Pour cela, on importe le module **random** :

```
In : from random import * ; help(random)
        random(...) method of random.Random instance
        random() -> x in the interval [0, 1).
                                                          python
```

3. A quoi sert l'instruction `random()` ?
4. On note $f : x \in [0, 1] \longmapsto \sqrt{1-x^2}$. Représenter la fonction f dans le plan muni d'un repère orthonormé, puis hachurer l'ensemble E défini par :
$$E = \{(x, y) \in [0, 1] \times [0, 1],\ y \leq \sqrt{1-x^2}\}$$

La **méthode de Monte-Carlo** consiste à choisir n points de coordonnées (x, y) au hasard avec $(x, y) \in [0, 1] \times [0, 1]$, puis pour les n points (x, y) obtenues, on itère les instructions suivantes :

- si $y \leq \sqrt{1-x^2}$, alors on ajoute 1 à un compteur c.
- sinon, le compteur c n'évolue pas.

On renvoie alors la fréquence de points appartenant à l'ensemble E définie par : $F = \dfrac{c}{n}$.

5. Dans le langage Python, construire le programme $montecarlo(n : int) \longrightarrow float$ qui pour tout entier n non nul donné, teste si n points choisis au hasard appartiennent à l'ensemble E, puis renvoie la fréquence de points appartenant à l'ensemble E. On a par exemple :

```
In : montecarlo(100)
                              0.76
                                                          python
```

6. Le changement de variable $t = \sin(u)$ nous donne rapidement $\displaystyle\int_0^1 \sqrt{1-t^2}\, dt = \frac{\pi}{4}$. Comment justifier que la méthode de Monte-Carlo semble aussi converger vers $\displaystyle\int_0^1 \sqrt{1-t^2}\, dt$?

Exercice 5 (calcul des termes de la suite de Fibonacci par mémoïsation). [ ]

On rappelle que la suite de Fibonnacci est définie par :

$$u_0 = 1, u_1 = 1 \text{ et pour tout } n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n$$

On cherche à comparer différents moyens de calculer la valeur de u_n. On rappelle par exemple qu'on peut utiliser un dictionnaire dans lequel on entre les résultats rencontrés. Concrètement,

- on ajoute en argument un dictionnaire vierge à la définition de la fonction,
- puis **avant** chaque appel récursif, on interroge le dictionnaire. Ainsi,
  - si le dictionnaire contient une valeur déjà calculée, le programme renvoie directement cette valeur,
  - si le dictionnaire ne contient pas la valeur, on la stocke dans le dictionnaire puis on continue l'exécution des instructions.

D'ailleurs, on rappelle que dans le langage Python, on définit un dictionnaire vide à l'aide des accolades {} :

```
In : notesmaths={}
```
python

Puis, on peut alors ajouter les couples clef-valeur k_i:v_i de la façon suivante, avant d'afficher le dictionnaire obtenu :

```
In : notesmaths['noah']=15,12,16; notesmaths['zoé']=14,12,14;
notesmaths['emignien']=16,15,11;

print(notesmaths)
```
{'noah': (15, 12, 16), 'zoé': (14, 12, 14), 'emignien': (16, 15, 11)}
python

Pour en extraire les valeurs, on renseigne simplement la clef utile. Par exemple, pour récupérer les notes de Noah :

```
In : notesmaths['noah']
```
(15, 12, 16)
python

1. Dans le langage Python, construire la fonction itérative $fibo1(n : int \longrightarrow int$ qui renvoie la valeur de u_n.
2. Dans le langage Python, construire la fonction récursive $fibo2(n : int) \longrightarrow int$ qui renvoie la valeur de u_n.
3. Construire alors le programme $fibo3(n : int, D : dict) \longrightarrow int$ qui renvoie la valeur de u_n, en utilisant un dictionnaire pour stocker les différentes valeurs rencontrées.
   Autrement dit, on veillera ici à construire un dictionnaire qui enregistre des couples de la forme n:u_n, puis on teste si n (et sa valeur) a déjà été rencontré, sinon on effectue le calul.
4. Construire le programme $comparaison(n : int) \longrightarrow None$ qui affiche sur un même graphe l'évolution du temps d'exécution des fonctions $fibo1$, $fibo2$ et $fibo3$ pour calculer $u_0, \ldots, u_n$.