Des suggestions d'organisation et de révisions en ITC

1 Remarques générales pour les écrits

- Avoir une rédaction suffisamment aérée ;
- Bien lire les questions et respecter le cahier des charges (une fonction qui renvoie un résultat doit se conclure par un `return`, un argument de type `list` qui ne doit pas être modifié doit être dupliqué de manière indépendante, ...) ;
- Bien matérialiser l'indentation de ses codes ;
- Dès qu'un code devient un peu élaboré, il faut le commenter ;
- Utiliser des noms de variables qui ne sont pas des fonctions natives en python ;
- Réfléchir à l'initialisation des variables ;
- Vérifier le nombre de passages dans les boucles (ce qui suppose une bonne connaissance des `range` dans la plupart des cas) ;
-  Penser à réutiliser les fonctions définies précédemment dans le sujet : cet aspect est très souvent négligé et sous-estimé alors qu'en suivant cette règle, le code gagne en lisibilité et aussi en fiabilité car on s'appuie sur une fonction écrite antérieurement supposée correcte ;
-  les questions `SQL` sont souvent très abordables (même s'il y a beaucoup de texte à lire avant de les aborder), il faut prendre ces points !

2 Balayage des items du programme

Sur les types :

- Connaissance des types simples (entiers, flottants, booléens) et composés (en particulier listes et dictionnaires) ;
- Écriture binaire d'un entier, méthode de complément à deux, représentation des flottants.

Des algorithmes classiques :

- Horner ;
- Exponentiation rapide (itératif ou récursif, avec complexité) ;
- Calcul d'un coefficient binomial (itératif ou récursif, avec complexité) ;
- Recherche dichotomique dans une liste triée (itératif, avec complexité).

Des algorithmes de tri (retenir le principe, ne pas apprendre le code par cœur) :

- Tri par insertion (avec complexité) ;
- Tri rapide pas en place (avec complexité) ;
- Tri fusion (avec complexité).

Sur les graphes :

- Notion de graphe orienté, non orienté, valué ;
- Représentation des graphes par listes d'adjacence ou matrice d'adjacence ;
- Principe du parcours en profondeur (récursif), parcours en largeur (avec une file) ;
- Recherche d'un plus court chemin, algorithme de Dijsktra et notamment l'opération de *relâchement*.

Sur les bases de données :

- Projection d'un attribut : `SELECT ... FROM ...` ;
- Limitation et décalage : `LIMIT, OFFSET` ;

— Suppression de doublons : `DISTINCT` ;
— Sélection : `WHERE` ;
— Fonctions d'aggrégations : `MAX, MIN, COUNT, SUM, AVG` ;
— Regroupement puis sélection : `GROUP BY ... HAVING ..` ;
— Requêtes avec sous-requêtes ;
— Jointures, jointures multiples : `... JOIN ... ON ..., ... JOIN ... JOIN ... ON ... AND ...`.

En programmation dynamique :
— Principe d'optimalité de Bellman (ou sous-structure optimale) : une solution optimale contient les solutions optimales des sous-problèmes (cas d'un plus court chemin, du rendu de monnaie, ...) ;
— Caractéristiques d'un problème d'optimisation pour une approche par programmation dynamique (sous-structure optimale, chevauchement des sous-problèmes) ;
— Approche descendante avec mémoïsation dans une liste ou un dictionnaire selon les cas ;
— Approche ascendante ;
— Connaissance d'un exemple comme le rendu de monnaie avec composition du rendu par un calcul ascendant (TP 10, exercice 2).

En théorie des jeux :
— algorithme de calcul d'un attracteur ;
— algorithme du minimax.

En ingénierie numérique :
— Méthode des rectangles ;
— Méthode d'Euler explicite ;
— Dichotomie, méthode de Newton.