Composition d'Informatique 2h - Filière PSI (XELCR)

1 Bilan

À titre liminaire, il est rappelé que le présent rapport ne concerne que la filière PSI et que seules les copies des candidats admissibles sont corrigées.

Cette année, 417 copies ont été corrigées. La moyenne s'établit à 7,2 et l'écart-type à 3,2. Toute l'échelle de notes a été utilisée, la note maximale s'établissant à 20 et la note minimale à 0,6.

Un total de 12 notes sont inférieures ou égales à 2 et sont donc susceptible d'entraîner l'élimination des candidats.

La répartition des notes est résumée dans le tableau suivant :

Intervalle de note	$[0, 4[$	$[4, 8[$	$[8, 12[$	$[12, 16[$	$[16, 20]$
Effectif	67 (16 %)	195 (47 %)	118 (28 %)	21 (8 %)	5 (1 %)

2 Généralités

Le sujet portait cette année sur un jeu de Tetris particulier, où toutes les pièces sont des barreaux de largeur 1 mais composés de blocs colorés dont le but est de provoquer l'alignement.

Une première partie avait pour but de familiariser les candidats à la manipulation de la grille composant le jeu (création, affichage). La seconde partie s'intéressait à des opérations simples sur celle-ci (apparition d'un barreau, descente, permutation, déplacement horizontal). La troisième partie proposait d'implémenter, par étapes, une méthode pour calculer le score. La quatrième partie traitait également du calcul du score, mais dans un contexte plus général où l'alignement était remplacé par la simple adjacence. Enfin, la cinquième partie était orientée vers la construction de requêtes SQL sur la base de données des scores.

Il est tout d'abord rappelé aux candidats que la maîtrise de la syntaxe du language python est indispensable pour cette épreuve. En particulier :

— L'indentation a un sens en python, et il est donc indispensable de la respecter.
— Python est sensible à la casse. Ainsi, `If` par exemple n'est pas un mot-clef python.
— Les listes sont indexées à partir de 0 et la fonction `range(k)` parcourt donc les valeurs de 0 à $k-1$. Cela signifie également que le dernier élément d'une liste L est accessible par l'instruction `L[len(L) - 1]`.

— La fonction `range(a, b, -1)` permet de parcourir les valeurs de l'ensemble $]b, a]$, de façon décroissante, mais la boucle n'itère donc pas si $b \geq a$.
— L'opérateur d'égalité est `==`, l'opérateur d'assignation est `=`.

Il est nécessaire de faire preuve d'une grande rigueur dans l'écriture d'un programme informatique et il ne faut pas hésiter à se relire pour corriger les éventuelles inattentions.

En outre, il importe de bien lire intégralement le sujet avant de commencer à composer, et en particulier l'introduction qui précisait les commandes autorisées. En l'espèce, il était explicitement requis que les candidats se limitent à l'utilisation de certaines commandes portant sur les listes, à l'exclusion de toutes autres. Ainsi, le jury a été amené à sanctionner :

— L'import de bibliothèques (en particulier `numpy` ou `copy`)
— L'utilisation de slices
— Les comparaisons, ajouts et multiplications de listes
— L'utilisation de l'opérateur binaire `in` (`if a in L`)
— L'utilisation d'indices négatifs dans les listes

Même en l'absence de limitations imposées, le jury conseille fortement aux candidats de se limiter aux syntaxes basiques de Python et à celles introduites dans le sujet. De nombreux candidats ont tenté d'utiliser les générateurs sans toujours connaître la syntaxe appropriée.

La clarté des programmes doit toujours être recherchée. Ainsi, il est conseillé aux candidats constatant une place insuffisante de passer à la page suivante, sachant qu'il est rare que soit demandé un programme de plus d'une demi-page pour une question donnée. Cela permettrait notamment d'éviter les ambiguïtés de niveau d'indentation causées par le changement de page.

La notation de Landau, rappelée dans l'introduction du sujet, n'est pas toujours comprise par les candidats. Ainsi, certains candidats n'ont pas compris qu'il leur était précisément demandé d'expliciter la fonction φ. En outre, les variables introduites doivent être définies (dire qu'une fonction est en $O(n)$ si n n'est pas défini n'a pas de sens, d'autant plus qu'ici largeur et hauteur ne sont pas interchangeables).

Enfin, il est toujours demandé aux candidats de fournir des fonctions qui n'entraînent pas d'erreurs, quelles que soient les valeurs de leurs paramètres, du moment qu'ils semblent plausibles. Ainsi, s'il n'est pas nécessaire de valider que la taille du barreau est positive ou que des coordonnées (x, y) en entrée sont dans la grille, le programme ne doit pas tomber en erreur si l'on demande de faire descendre un barreau déjà au fond de la grille ou de décaler un barreau qui est bloqué par le bord de la grille ou un autre barreau. À ce sujet, il est rappelé que les conditions composées à l'aide des opérateurs `or` ou `and` sont évaluées (si nécessaire) de gauche à droite. Ainsi, `a < len(grille) and grille[a]` ne provoque pas d'erreur si a est trop grand, mais `grille[a] and a < len(grille)` le fait.

3 Commentaires détaillés

3.1 Question 1

0	$]0, 0{,}5[$	$[0{,}5, 1[$	1
81 (19 %)	92 (22 %)	79 (19 %)	165 (40 %)

Il s'agissait de créer un tableau à deux dimensions (`largeur` et `hauteur` données en argument), ou plus précisément un tableau de taille `largeur` dont chaque élément est lui-même un tableau de taille `hauteur` et dont tous les éléments contiennent la valeur `VIDE`. Outre les petites erreurs de syntaxe déjà mentionnées en préambule, il fallait être attentif à ne pas inverser `largeur` et `hauteur`, à bien initialiser les entrées du tableau à la valeur `VIDE` (et non 0 par exemple), et surtout à ne pas réutiliser un tableau (de taille `hauteur`) déjà créé.

3.2 Question 2

0	]0, 0,5[	[0,5, 1[	1
64 (15 %)	68 (16 %)	205 (49 %)	80 (19 %)

Cette question demandait d'afficher le contenu du tableau à l'écran, à l'aide de 3 fonctions auxiliaires. Même si le sujet attirait l'attention dessus, la principale difficulté est restée de parcourir le tableau dans le bon sens, de nombreux candidats ne procédant qu'à une des deux inversions nécessaires.

3.3 Question 3

0	]0, 0,5[	[0,5, 1[	1
107 (26 %)	89 (21 %)	157 (38 %)	64 (15 %)

Cette question consistait à parcourir un sous-tableau de taille `largeur * k` pour vérifier si une des ses colonnes ne contient que des cases vides. De nombreux candidats ont bien fait appel à deux boucles imbriquées pour parcourir le sous-tableau, mais n'ont pas toujours bien défini les bornes des indices des cases à parcourir. D'autre part, si la valeur `k` était donnée en argument de la fonction, la valeur `largeur` ne l'était pas et devait donc être calculée.

3.4 Question 4

0	]0, 0,5[	[0,5, 1[	1
52 (12 %)	68 (16 %)	221 (53 %)	76 (18 %)

Il s'agissait ici de faire descendre le barreau d'une case. Certains candidats ont mal lu l'énoncé et compris qu'il fallait plutôt faire descendre le barreau autant que possible, ce qui n'était pas correct. Pour les autres, il fallait évidemment penser à vérifier que le barreau n'était pas bloqué, que ce soit par le bas de la grille ou un autre bloc, et ne pas oublier d'effacer la case libérée.

3.5 Question 5

0	]0, 0,5[	[0,5, 1[	1
50 (12 %)	81 (19 %)	215 (52 %)	71 (17 %)

Cette question demandait de déplacer le barreau horizontalement dans la grille, en vérifiant au préalable que ce déplacement était possible. Cette vérification a entraîné son lot d'erreurs, la plus fréquente étant d'accéder à des cases en dehors du tableau représentant la grille.

3.6 Question 6

0	]0, 0,5[	[0,5, 1[	1
145 (35 %)	25 (6 %)	127 (30 %)	120 (29 %)

Il fallait ici opérer une permutation des couleurs au sein même du barreau. Si la plupart des candidats ont identifié que la dernière case était un cas particulier, la principale difficulté était de ne pas copier la même couleur sur chaque autre case, ce qui était probable avec une solution trop naïve. Plusieurs solutions étaient possibles pour l'éviter (parcours de haut en bas ou échange de cases). Il était certainement utile de s'aider d'un exemple au brouillon ou sur la copie.

3.7 Question 7

0	]0, 0,5[	[0,5, 1[	1
162 (39 %)	79 (19 %)	143 (34 %)	33 (8 %)

La grande majorité des candidats a tenté de proposer une solution de la complexité demandée (en $O(\texttt{k} + \texttt{hauteur})$), mais beaucoup ont eu des difficulté à calculer les coordonnées de la première case non-vide sous le barreau, le plus souvent à cause d'une erreur dans la condition d'arrêt de leur boucle, provoquant une erreur dans le calcul de coordonnées, ou plus grave, un accès au tableau représentant la grille au-delà de ses bornes. Ensuite, il fallait réaliser le déplacement du barreau et prendre garde à bien traiter le cas particulier où le barreau ne se déplace pas : dans ce cas, copier le contenu du barreau à sa nouvelle position, puis remplacer par `VIDE` les cases aux anciennes positions revient à effacer le barreau, ce qui n'est pas correct.

3.8 Question 8

0	]0, 0,5[	[0,5, 1[	1
132 (32 %)	2 (0 %)	155 (37 %)	128 (31 %)

Cette question d'application était relativement facile, mais nécessitait une grande rigueur dans la lecture du sujet comme dans l'application. Il était en effet indispensable de prendre le temps de représenter l'état intermédiaire de la grille après le premier tassement, sans se tromper. En outre, il fallait bien lire qu'un même bloc pouvait servir simultanément dans plusieurs alignements.

3.9 Question 9

0	]0, 0,5[	[0,5, 1[	1
256 (61 %)	71 (17 %)	74 (18 %)	15 (4 %)

Cette question a été réussie par un très petit nombre de candidats. Outre les petites erreurs dans le calcul d'indices et les conditions d'arrêt des boucles, des erreurs plus grossières ont été maintes fois constatées, comme le fait de traiter la valeur `VIDE` comme si c'était une couleur (c.-à-d. comptabiliser les alignements de cases vides), ou ne pas bien traiter la fin du tableau (par exemple, oublier qu'arriver à la fin du tableau a essentiellement le même effet qu'un changement de couleur).

3.10 Question 10

0	]0, 0,5[	[0,5, 1[	1
143 (34 %)	43 (10 %)	170 (41 %)	61 (15 %)

Cette question était plus facile et consistait à appliquer la fonction précédente à un alignement donné de la grille. Si la construction de la rangée a été en général bien faite, les candidats n'ont pas tous su établir les diverses conditions d'arrêt pour éviter de sortir de la grille. En outre, il fallait faire attention à ne pas modifier plus que nécessaire la grille `g`, donnée en paramètre, et donc bien lire le sujet à cet égard. En particulier, ce n'était pas à la fonction d'y copier le contenu de `grille`.

3.11 Question 11

0	]0, 0,5[	[0,5, 1[	1
330 (79 %)	30 (7 %)	49 (12 %)	8 (2 %)

Relativement peu de candidats ont traité cette question qui consistait à balayer les lignes, colones et diagonales de la grille à la recherche d'alignements unicolores.

Les candidats qui ont abordé cette question ont souvent pu traiter avec succès les lignes et les colonnes, mais ont plus souvent manqué le balayage des diagonales, oubliant d'en couvrir certaines ou explorant plusieurs fois les mêmes.

3.12 Question 12

0	]0, 0,5[	[0,5, 1[	1
277 (66 %)	49 (12 %)	85 (20 %)	6 (1 %)

Il était possible de fournir une solution très simple à cette question en réutilisant la fonction `descenteRapide(grille, x, y, k)` avec un barreau de taille 1. Toutefois, la complexité obtenue n'était alors pas optimale. Il était donc plutôt nécessaire de construire une solution similaire mais évitant les redondances entre calcul de la taille du barreau, de l'espace libre en dessous et de la prochaine case à considérer après la descente.

3.13 Question 13

0	]0, 0,5[	[0,5, 1[	1
261 (63 %)	16 (4 %)	95 (23 %)	45 (11 %)

Il fallait simplement itérer l'appel à `effaceAlignement(grille)` (Question 11) et `tassementGrille(grille)` (Question 12) pour calculer le score total.

Si peu de candidats ont abordé cette question plutôt facile, probablement par manque de temps, ceux qui l'ont fait l'ont assez bien réussie. Il n'y avait pas de grande difficulté et les erreurs constatées s'apparentent à des distractions, comme l'oubli d'appeler la procédure de tassement de la grille, la mauvaise gestion du tuple renvoyé par la procédure `effaceAlignement(grille)`, ou une condition d'arrêt de boucle incorrecte.

3.14 Question 14

0	]0, 0,5[	[0,5, 1[	1
402 (96 %)	10 (2 %)	5 (1 %)	0 (0 %)

Cette question demandait la production d'un algorithme récursif, dont la grande difficulté était de définir les conditions d'arrêt. Celui-ci reposait nécessairement sur la modification de `grille` pour vider la case courante, qu'il fallait gérer au bon moment pour éviter que la récursion puisse revenir sur une case déjà traitée. Il était donc nécessaire d'avoir une bonne compréhension des conséquences de la modification d'une liste passée en argument.

Malgré quelques bonnes réponses, aucun candidat n'a obtenu l'intégralité des points.

3.15 Question 15

0	]0, 0,5[	[0,5, 1[	1
409 (98 %)	1 (0 %)	6 (1 %)	1 (0 %)

Extrêmement peu de candidats ont traité cette question, qui n'était pas très difficile, mais demandait un important investissement en temps.

Le code à analyser n'était pas correct, ce qui pouvait être démontré en donnant un exemple de grille et une justification succinte.

3.16 Question 16

0	$]0,0{,}5[$	$[0{,}5,1[$	1
156 (37 %)	24 (6 %)	161 (39 %)	76 (18 %)

Cette question, et de façon plus générale la partie 5, a été relativement bien traitée par les candidats, beaucoup identifiant à juste titre sa difficulté moindre et son indépendance vis-à-vis des questions précédentes.

La plupart des erreurs dérivent d'une méconnaissance de la syntaxe du langage SQL, et en particulier des jointures.

3.17 Question 17

0	$]0,0{,}5[$	$[0{,}5,1[$	1
243 (58 %)	29 (7 %)	84 (20 %)	61 (15 %)

Sans difficulté particulière, cette question a été assez bien réussie par les candidats qui l'ont traitée. Les erreurs courantes ont été d'oublier de faire $+1$ après comptage des scores supérieurs à s, ou de compter les scores $\geq s$ au lieu de $> s$.

3.18 Question 18

0	$]0,0{,}5[$	$[0{,}5,1[$	1
178 (43 %)	17 (4 %)	119 (29 %)	103 (25 %)

Cette question était finalement très similaire à la question 16 et nécessitait simplement de connaître la fonction d'agrégation `MAX`. Certains candidats ont pu pallier cette méconnaissance en utilisant une combinaison des instructions `ORDER BY` et `LIMIT`.

3.19 Question 19

0	$]0,0{,}5[$	$[0{,}5,1[$	1
313 (75 %)	41 (10 %)	57 (14 %)	6 (1 %)

Cette question nécessitait de concevoir une requête SQL imbriquée, aspect pour lequel le jury a été indulgent, et de ne compter chaque joueur qu'une seule fois (par exemple en utilisant le mot-clé `DISTINCT`). Assez peu de candidats ont proposé une réponse qui soit proche de la solution correcte.