ÉCOLE POLYTECHNIQUE CONCOURS D'ADMISSION 2019
ENS : LYON, PARIS-SACLAY, RENNES

Rapport sur la composition d'informatique 2h, filière MP (XLCR)

Bilan général *(Candidats français)*

0<=N<4	22	11,7%
4<=N<8	94	50%
8<=N<12	64	34,04%
12<=N<16	8	4,26%
16<=N<=20	0	0
Total :	188	100%
Nombre de candidats :	188	
Note moyenne :	7,15	
Ecart-type :	2,67	

Commentaires

Cette année, le sujet portait sur la réalisation d'un jeu de type Tetris, où un joueur fait tomber un barreau coloré dans une grille, dans le but de réaliser des alignements monochromes horizontaux les plus grandes possibles. La première partie portait sur l'initialisation et l'affichage d'une grille. La seconde partie se concentrait sur les déplacements possibles du barreau. Dans la troisième partie il était demandé d'implémenter la détection d'alignements de couleurs et le calcul du score. La quatrième partie considérait une variante plus technique, où les régions monochromes étaient décomptées. Enfin, la cinquième partie étudiait la gestion d'une base de données regroupant les scores des parties effectuées.

Le langage Python utilisant l'indentation comme marqueur de blocs, il est rappelé -- comme chaque année -- de bien veiller à ce que cette indentation soit la plus claire possible: soit en suivant scrupuleusement le quadrillage de la copie (compter au moins 1cm par indentation), soit en traçant des repères verticaux au niveau de chaque tabulation pour lever tout doute. Attention dans ce cas : bien veiller à ne pas omettre de bloc, et à ne pas inclure la ligne "d'en-tête" du bloc, qui ne doit pas être indentée.

L'énoncé décrivait précisément les opérations autorisées sur les listes : création de liste vide, `len`, `append`, accès via `L[i]` pour 0 ≤ i < len(L), et compréhension de liste `[ … for x in …. ]`. Les opérations non-autorisées incluaient donc:

- les tranches de listes `L[i:j]`
- la copie de liste via `list.copy` ou `L[:]` (qui par ailleurs étaient souvent mal comprises dans les tableaux à deux dimensions)
- les accès par le dernier élément `L[-1]`
- la multiplication `*` pour recopier une liste

- l'égalité entre les listes

De nombreuses copies ont fait preuve d'inattention avec les éléments basiques du langage: parenthèses aux appels de fonction ; deux-points en fin de `if`, `for`, ou `while` ; oubli de l'opérateur `*` pour la multiplication ; `==` pour la comparaison ; distinction majuscule/minuscule (en particulier pour la constante "`VIDE`"), etc. Ces erreurs traduisent souvent une difficulté à écrire un code correct de soi-même, sans avoir à se reposer sur un éditeur ou la sortie du compilateur qui énumèrent les imprécisions.

Une autre source de confusion portait sur les paramètres et valeurs de retour des fonctions et procédures. La plupart travaillaient sur une grille passée en paramètre, et il était parfois demandé de modifier la grille en place, ou bien de travailler sur une copie et de la renvoyer: une mauvaise compréhension de la consigne entraînait au mieux des `return` superflus, au pire une destruction de la grille originale.

Il était rappelé l'utilité des commentaires, en particulier en introduction de fonctions, pour clarifier le code et l'algorithme employé. Certains sont tombés dans l'excès inverse, en paraphrasant l'intégralité de leur code dans un long commentaire en début de chaque question: il est préférable de focaliser les commentaires sur les points essentiels.

Enfin, on rencontre régulièrement des instruction redondantes, du type "`if a == true:`", ou bien un `else if` qui reprend exactement la négation de la première condition. Cela ne représente pas des erreurs à proprement parler, mais traduisent un manque de recul sur le code que l'on est en train d'écrire.

Commentaires détaillés

Pour chacune des questions, sont indiquées les statistiques suivantes:

- Traitement: taux de traitement de la question par les candidats (les autres lignes se rapportent uniquement aux copies ayant traité la question)
- Moyenne: note moyenne pour cette question, ramenée sur 1
- 0,]0 ; 0,5[, [0,5; 1[, 1 : pourcentage de copies dont la note est dans l'intervalle correspondant

Partie 1

Cette partie avait pour vocation de familiariser le candidat avec les grilles manipulées, via la réalisation de deux fonctions simples (initialisation et affichage de la grille). Pour ces fonctions simples, il était important de bien veiller à respecter les bases du langage.

Question 1

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
100 %	0,65	34 %	0 %	1 %	65 %

L'erreur la plus fréquente consistait à réutiliser une même ligne répétée `hauteur` fois (ces fonctions avaient généralement une complexité en O(hauteur+largeur), alors qu'il y a hauteur*largeur cases à créer, et que les fonctions autorisées ne permettaient de créer les cases qu'une par une. Certains candidats ont également inversé les axes de la grille. La

solution la plus efficace (et la plus courte) consistait à utiliser deux compréhensions de listes imbriquées.

Question 2

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
99 %	0,54	44 %	0 %	3 %	53 %

Il s'agissait de réaliser deux boucles for imbriquées: une boucle pour les lignes contenant une boucle pour les colonnes, affichant les cases une à une. Il fallait être attentif aux indices, et à l'ordre vertical, l'ordre des lignes de la grille étant inversé par rapport à l'ordre des lignes des fonctions d'impression à l'écran.

Partie 2

Question 3

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
100 %	0,68	15 %	4 %	31 %	50 %

Cette question ne posait pas de piège particulière (en dehors d'un besoin de précision syntaxique). La solution à cette question pouvait être simplifiée par l'utilisation d'une fonction annexe qui teste si les cases sont libres à des coordonnées données (une telle fonction pouvait être réutilisée dans la question 5). Le calcul de la complexité a régulièrement été oublié alors qu'il était correct pour la grande majorité des candidats qui l'ont réalisé.

Question 4

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
100 %	0,66	16 %	3 %	35 %	46 %

Deux vérifications étaient attendues avant de réaliser la descente: y>0 d'une part, et que la case en dessous était libre d'autre part (l'ordre des vérifications ayant une importance). Après avoir recopié les éléments un par un, il importait d'effacer l'élément du haut.

Question 5

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
100 %	0,74	9 %	3 %	32 %	55 %

Pour cette question il fallait vérifier que la colonne de destination (x+direction) était dans le bon intervalle et contenait bien des cases libres. Beaucoup de candidats ont perdu du temps en traitant les deux directions séparément, au lieu de simplement recopier les cases de la colonne x vers la colonne s+direction.

Question 6

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
100 %	0,54	44 %	0 %	4 %	52 %

Deux approches étaient possibles pour cette question. La première consiste à sauvegarder la couleur du barreau supérieur, recopier chaque couleur une case plus haut (de haut en bas), puis réinsérer la sauvegarde en bas. La seconde consiste à exécuter des inversions deux à deux, soit avec des éléments adjacents (de haut en bas), soit entre l'élément supérieur et chaque autre élément successivement, de bas en haut.

Question 7

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
99 %	0,57	8 %	28 %	40 %	25 %

Cette fonction supposait de d'abord rechercher la case de destination, avant de réaliser la descente du barreau sans oublier d'effacer l'original. La recherche de la première case non-vide devait se faire de haut en bas, pour gérer le cas où la grille n'est pas tassée. Deux autres cas spécifiques ont souvent été oubliés: la colonne est entièrement vide (la recherche de la première case non-vide se poursuit alors dans les lignes négatives), ou bien il n'y a pas de case libre, auquel cas la destination est identique au point de départ, et l'effacement de l'original se solde souvent par l'effacement complet du barreau.

Partie 3

Question 8

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
99 %	0,43	50 %	0 %	13 %	36 %

Cette question en apparence simple demandait une attention toute particulière pour ne rater aucun alignement (pour un total de 8 points). La grande majorité des erreurs sont dues à des fautes d'inattention, plutôt qu'à une mauvaise compréhension de l'énoncé. La justification la plus efficace consistait à dessiner les grilles successives après chaque tassement, en identifiant bien les alignements réalisés.

Question 9

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
96 %	0,50	15 %	22 %	49 %	15 %

Il était possible de réaliser cette question en une passe (recherche des changements de couleurs et calcul du score à la volée), ou bien deux (enregistrement des différences de couleurs dans un tableau, puis parcours de ce tableau). Les deux principaux écueils étaient la gestion de la dernière séquence (qui devait généralement être traitée à part, à l'extérieur de la boucle), et des séquences vides (qui étaient trop souvent comptées comme des séquences à part entière).

Question 10

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
89 %	0,68	8 %	22 %	26 %	44 %

Une erreur presque universelle pour cette question était une mauvaise gestion du cas dx=dy=0: dans la grande majorité des cas ces paramètres menaient à une boucle infinie.

Question 11

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
81 %	0,41	26 %	27 %	36 %	11 %

La principale difficulté était de s'assurer de parcourir toutes les lignes, les colonnes et les diagonales sans oubli ni répétition. Par ailleurs, il était demandé de ne pas modifier la grille passée en paramètre ; la première étape consistait donc à en faire une copie profonde, ce qui a posé quelques difficultés aux candidats. Remarque: même si l'opérateur de copie "[:]" avait été autorisé, grille[:][:] ne permet pas de faire une copie complète d'un tableau à deux dimensions. La compréhension de liste est la solution la plus simple pour ce problème.

Question 12

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
67 %	0,54	15 %	22 %	47 %	16 %

Le tassement d'une colonne pouvait se faire en la remplissant avec les cases non vides rencontrées lors du parcours effectué en remontant depuis le bas. Les tentatives de réutiliser descente ou descenteRapide menaient au mieux à des solutions de complexité sous-optimale.

Question 13

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
64 %	0,53	18 %	10 %	44 %	27 %

Cette question ne posait pas de difficulté particulière ; il fallait veiller à renvoyer la somme des scores incrémentaux renvoyés par effaceAlignement lors des appels alternés à effaceAlignement et tassementGrille.

Partie 4

Question 14

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
38 %	0,47	29 %	17 %	33 %	20 %

Pour proposer une solution récursive, il était nécessaire de modifier la grille en effaçant les cases participant d'une région unicolore lors du parcours. La justification de la terminaison, bien qu'elle ne posait pas de difficulté, n'a pas toujours été fournie. Comme c'est souvent le cas avec les fonctions récursives, certains candidats ont tenté des solutions

inutilement compliquées où chaque appel récursif était précédé d'un grand nombre de tests. Ici, faire les vérifications sur les coordonnées et les couleurs en début de fonction augmentait grandement la simplicité de la solution.

Question 15

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
13 %	0,39	40 %	21 %	15 %	24 %

La fonction exploreRegion présentait plusieurs écueils, que l'on pouvait mettre en lumière à l'aide d'un contre-exemple. Il fallait veiller à bien spécifier le contre-exemple (grille et case de départ).

Partie 5

Question 16

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
68 %	0,85	7 %	2 %	13 %	78 %

Il fallait utiliser une jointure, et ne pas oublier de trier les résultats par ordre chronologique (ORDER BY).

Question 17

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
53 %	0,76	5 %	11 %	29 %	55 %

L'erreur la plus fréquente a été un décalage de 1 dans le rang.

Question 18

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
55 %	0,88	4 %	7 %	12 %	77 %

Cette question ne posait pas de difficulté particulière, il fallait faire une jointure et faire appel à la fonction MAX().

Question 19

Traitement	Moyenne	0	]0;0.5[	[0.5;1[	1
38 %	0,59	12 %	23 %	35 %	30 %

On pouvait ici utiliser une requête auxiliaire (qui correspondait en réalité à la requête de la question précédente).