Le candidat idéal capable de répondre à toutes ses exigences a été rencontré, mais force est de constater que plusieurs candidats présentent des difficultés dans l'analyse d'une courbe de titrage, la gestion du temps et l'organisation de la paillasse.

Ce concours est une épreuve où l'énergie, la ténacité et le sens de l'initiative sont demandés, le jury ne pénalisera pas les éventuelles questions des candidats et y répondra si possible.

4. INFORMATIQUE

4.1. Informatique pour tous

- Remarques générales

Le sujet d'informatique commune portait sur la modélisation d'une situation de trafic routier. Le sujet balayait un spectre assez large des notions abordées lors des deux années de CPGE.

Le sujet, assez long, mais d'une difficulté raisonnable, a permis un classement très efficace des candidats, tout le spectre de notes possibles étant occupé. Certaines copies nous impressionnent favorablement par leur maîtrise et leur rapidité, d'autres nous impressionnent beaucoup moins favorablement.

Les clefs pour obtenir une note honorable à cette épreuve sont de bien maîtriser la syntaxe de base de Python et du SQL de s'assurer que les programmes font exactement - et pas « à peu près » - ce qui est demandé par l'énoncé, et de proposer des programmes concis et bien présentés/indentés.

Cette année, le jury croit utile d'attirer l'attention des futurs candidats et de leurs formateurs sur les points suivants :

- Un défaut particulièrement sensible cette année a été le **manque de concision des réponses et des programmes proposés, même pour des fonctions élémentaires**. Nous avons trop souvent été confrontés à des programmes surréalistes pour tester par exemple si une case est occupée ou non, ou bien pour vérifier que deux listes sont égales... mais nous avons trop rarement vu des programmes simples, qui étaient pourtant parfaitement accessibles grâce aux notions du programme (voir les commentaires de la question Q3, symptomatique de ce problème). Quant aux questions sur l'évolution de la file de voitures, on a parfois dû subir des programmes de 20 lignes ou plus avec des disjonctions de cas inutiles alors que 2 lignes suffisaient en maîtrisant la syntaxe de base de Python. Notre message principal aux candidats sera donc, concernant les programmes qu'ils proposent :

  ALLEZ AU PLUS SIMPLE !

- Dans le même esprit, le jury demande explicitement aux candidats d'éviter de dépenser leur précieux temps d'épreuve à commenter leurs programmes, tout particulièrement les programmes relativement simples. Ce qui est parfaitement compréhensible et souhaitable avec des programmes complexes dans un cadre industriel ou de recherche ne l'est plus dans le cadre contraint d'une épreuve de concours en temps limité. La perte de temps qui résulte de commentaires superflus a pénalisé de nombreux candidats : nous tenions donc à faire cette mise au point.

- Concernant le détail de la syntaxe, de nombreux candidats accordent une attention trop limitée aux bornes des itérateurs. Toute erreur sur ce point est sanctionnée, notamment sur les questions simples où il n'est pas acceptable de lire **for i in range(len(L)+1) :** pour parcourir une liste de longueur **len(L)**. Attention également à **range(p,q,-1)** souvent mal maîtrisé (on rappelle que **list(range(p,q,-1))** renvoie **[p,p-1,… ,q+1]** et qu'il est nécessaire que **p>q** si on veut une liste non vide).

- Une faute grave a été vue dans un nombre appréciable de copies : l'utilisation de **while** à la place de **if**. Cette confusion est sanctionnée lourdement.

- Une erreur de base vue un certain nombre de fois concerne la gestion des listes et de l'affectation des éléments. On a lu de trop nombreuses fois des instructions telles que :
  L=[]
  L[0]=b
  qui ne fonctionne évidemment pas.
  Quant à l'ajout d'un élément en fin de liste, nous avons trop souvent vu **L=L+a** à la place de **L=L+[a]**. De plus, trop de candidats pensent à tort que **L.append(L1)** est équivalent à **L+L1 :** seule cette dernière est correcte.

- Les candidats doivent savoir que l'instruction **L2=L** ne suffit pas à créer une liste **L2** indépendante de **L**.
- Enfin, sur la présentation, le jury insiste tout particulièrement sur la lisibilité de l'écriture - il arrive quelquefois que nous renoncions à tenter de déchiffrer une réponse si elle est très mal écrite - et sur la clarté de l'indentation (les lignes verticales de la feuille sont souvent largement suffisantes si le candidat les utilise bien), celle-ci étant cruciale pour vérifier la validité des programmes.

- Commentaires par question

Q 1 : Rappelons aux candidats que 0 et 1 ne sont pas des valeurs de variable booléenne.
Q 2 : Que de réponses compliquées ou même fausses sur cette question élémentaire ! En particulier, de nombreux candidats pensent que définir une fonction suffit à créer une liste. Ce n'est le cas que si on appelle la fonction…
Q 3 : La solution la plus simple, **def occupe(L,i) : return L[i]** n'a été proposée que par une minorité de candidats. De très nombreux candidats proposent :

```
def occupe(L,i) :
    if L[i]==True :
        return True
    elif L[i]==False :
        return False
```

certes correcte, mais qui montre qu'il y a encore une compréhension imparfaite de la puissance du langage. Parmi les réponses fausses, on a souvent vu une boucle for complètement hors de propos, qui nous montrait dès la troisième question que le candidat ne comprenait pas bien ce qu'il faisait.
Q 4 : Question en général correctement traitée, mais qui a donné lieu à des réponses parfois surprenantes… on a ainsi vu des candidats faisant de complexes raisonnement de dénombrement, allant même parfois jusqu'à un raisonnement par récurrence, qui s'assimilait ici à une perte de temps au vu de la simplicité de la question et de la justification attendue.

Q 5 : Parmi les fautes récurrentes relevées sur cette question, un **return True** mal placé qui faisait sortir de la boucle avant d'avoir testé tous les éléments, une double boucle **for** inutile, et quelques (heureusement rares) candidats qui semblent penser que deux listes de mêmes longueurs sont forcément égales…
Certains utilisent volontiers **assert len(L1)==len(L2)** en début de programme. Cela sert au débogage d'un programme en renvoyant une exception (notion hors programme), mais ne renvoie pas un booléen comme demandé.
Q 6 : Question bien traitée en général, même si on a vu quelques complexités curieuses ou irréalistes.
Q 7 : Question assez mal traitée au regard de sa faible difficulté. Le sujet de l'an dernier proposait pourtant une question similaire. Nous rappelons aux candidats que « True ou False » n'est pas un type de variable.
Q 8 : De nombreux candidats ont répondu par un dessin. Cette fonction ne renvoyait pas un dessin, mais une liste de booléens.
QQ 9 -10 : Certains candidats proposent des réponses exactes et concises. À l'inverse, d'autres proposent des solutions extrêmement complexes, avec des indices dans les itérateurs et les listes difficiles à interpréter. On passe sur les quelques candidats qui ont fait reculer les voitures au lieu de les faire avancer... Dans ces deux questions, il fallait créer une nouvelle liste indépendante de **L** (**L2=L** ne suffisait donc pas), et faire bien attention aux indices dans les découpages de listes (non, le dernier élément de **L[0:m]** n'est pas **L[m]**, et **L[len(L)]** n'existe pas…).
Q 11 : Question plutôt difficile que certains bons candidats ont remarquablement traitée. Même si on percevait la nécessité de rechercher la première case accessible et faire avancer les voitures situées avant, il fallait également faire attention à la gestion du booléen b : le premier élément de la liste ne prenait pas automatiquement la valeur de b.
Q 12 : Pour traiter cette question de manière raisonnable, il fallait bien entendu judicieusement utiliser les fonctions définies dans les questions 9, 10 et 11. Les candidats qui ont voulu faire autrement se sont souvent embourbés dans des programmes trop complexes, ou dans des disjonctions de cas déraisonnables.
Q 13 : Question assez simple, attention toutefois à la syntaxe correcte de la réponse : cette fonction renvoie non pas deux listes, mais une seule liste constituée de deux sous-listes.
Q 14 : Question qualitative bien traitée.
Q 15 : De nombreux candidats ont répondu 8 étapes ou 10 étapes en allant trop vite.
Q 16 : Le jury apprécie des réponses claires et non ambiguës sur ces questions qualitatives. Le doute ne bénéficie pas au candidat.
Q 17 : Question en apparence simple, mais qui contenait de nombreux pièges dans lesquels beaucoup de candidats sont tombés. La précision de l'énoncé « de complexité linéaire » a échappé à beaucoup, y compris à ceux qui pensent s'en sortir en utilisant **in**, **del** ou **.pop** ou… qui ne sont certainement pas de complexité unitaire ! Attention également à la gestion des indices : on a souvent vu des fonctions qui renvoyaient une liste ne contenant pas le premier ou le dernier élément de la liste primaire.
QQ 18-19 : Questions assez bien traitées.
Q 20 : Un certain nombre de candidats ont vu en but un entier, ce n'était pas le cas.
Q 21 : Pour obtenir tous les points sur cette question, il fallait clairement reconnaître un algorithme de recherche par dichotomie et citer les complexités des deux algorithmes proposés. Les rares candidats qui ont fait valoir que « **in1** est plus optimisé, car il y a moins d'instructions » nous ont laissés songeurs.
Q 22 : Quelques candidats ont traité correctement cette question, mais il fallait faire attention aux puissances de 2 employées.
Q 23 : Question rarement traitée.
Q 24 : Trop de candidats écrivent « il y a un **while** donc la boucle s'arrête nécessairement »… encore faut-il le prouver !

Q 25 : Question rarement traitée, mais souvent de manière satisfaisante.
Q 26 : Cette requête simple, qui ne nécessitait pas de jointure, a été relativement mal traitée.
Q 27 : Les candidats semblent maîtriser plutôt mieux que l'an dernier la syntaxe d'une jointure, mais ne maîtrisent pas encore assez la notion : combien de fois a-t-on vu **ON Croisement.id=Voie.id** dans la condition de jointure ?
Q 28 : Question assez peu traitée.

Quelques distinctions honorifiques :
Prix « rencontre du troisième type » :
« Cette fonction renvoie un booléen qui est une chaîne de caractère de type str. »
Prix de l'extrapolation douteuse :
« Les voitures se sont montées dessus dans l'intersection. »
Prix Jean-Claude Van Damme :
« Il n'existe qu'une seule file de longueur n car toute file de longueur n est contenue dans elle-même. »

4.2. Informatique — filière MP

- Remarques générales

Le sujet est constitué d'un problème en deux parties : la première partie traite des automates et la seconde porte sur la programmation. L'ensemble permet de bien évaluer l'acquisition du programme des deux années de classe préparatoire.
Les candidats abordent les deux parties dans leur grande majorité. Ils finissent parfois le sujet (en passant les questions difficiles). Quelques (rares) excellentes copies ont pu être lues.
La présentation des copies est globalement satisfaisante.
Nous avons pu constater peu d'efforts de rédaction des questions théoriques (automate ou calcul de complexité).
Beaucoup de candidats ne donnent pas d'arguments ou se contentent d'arguments superficiels.

- Partie automate

Le but de cette partie est de manipuler les automates en relation avec une certaine classe de langages.
Le sujet permet d'évaluer les connaissances des candidats en matière d'automates et leur capacité à identifier le langage accepté par un automate.
Des exemples sont traités dans un premier temps. Ceci permet de mettre en place une approche formelle dans le cas général.
La difficulté est essentiellement dans les justifications formelles attendues pour certaines questions.

- Quelques remarques

QQ 1-6
De très rares erreurs. L'association langage-automate est bien maîtrisée.
QQ 7-8
Des erreurs plus fréquentes sans doute dues à une mauvaise interprétation des opérations ensemblistes utilisées.
QQ 9-12
Ces questions nécessitent une argumentation formelle fine et rigoureuse.