Epreuve orale d'Informatique, Filière PT

L'épreuve orale d'informatique a pour but de tester les connaissances des candidats sur le programme officiel d'informatique (première et seconde années) pour la filière PT.

L'épreuve se déroule de la façon suivante. Le candidat dispose d'une heure de préparation sur un sujet donné par l'examinateur. Une fois cette heure écoulée le candidat présente pendant quarante-cinq minutes ses résultats devant l'examinateur qui lui demande des précisions et lui pose des questions ou des exercices complémentaires.

À aucun moment de l'épreuve nous n'utilisons un ordinateur, il ne s'agit donc absolument pas d'une épreuve sur machine. Cependant, les examinateurs sont particulièrement attentifs à la capacité du candidat à écrire un programme syntaxiquement correct dans le langage Python. Ainsi, le candidat doit être capable pour chaque question demandant l'écriture d'un programme de proposer un code Python valide.

Vingt-trois candidats ont passé l'épreuve. Le jury regrette qu'il n'y ait pas eu de candidate féminine admissible.

La moyenne a été de 12,13/20 et l'écart type de 3,22. Les notes se sont réparties comme suit :

- 1 candidat a obtenu une note inférieure ou égale à 6
- 5 candidats ont obtenu une note comprise entre 7 et 9
- 4 candidats ont obtenu une note comprise entre 9,5 et 11
- 7 candidats ont obtenu une note comprise entre 12 et 14
- 6 candidats ont obtenu une note supérieure ou égale à 15

Les sujets comportaient systématiquement des questions de nature algorithmique. Certaines portaient directement sur des algorithmes classiques vus en cours (par exemple des algorithmes sur les listes — recherche, tri — ou de l'algorithmique des graphes) mais il y avait également des questions demandant aux candidats d'élaborer des algorithmes plus originaux.

La grande majorité des candidats a démontré une bonne connaissance du programme et le jury a eu le plaisir de voir certains candidats faire preuve d'une très grande créativité.

Il n'y a pas eu à déplorer de difficulté dans l'écriture de code Python syntaxiquement correct.

Cette épreuve comportant une longue préparation (une heure) nous recommandons aux candidats d'en profiter pour bien rédiger en amont leurs réponses afin de ne pas se retrouver à devoir développer en direct une idée (même correcte) trop peu détaillée. Les questions sont traitées dans l'ordre par les examinateurs qui n'hésitent pas à passer le temps nécessaire avant de passer à la question suivante : il est donc illusoire d'espérer que l'on sautera une question non traitée par le candidat même si les examinateurs prennent par ailleurs soin d'aborder lors de l'interrogation orale le plus de questions possible que le candidat a eu le temps de préparer (ainsi, au début de l'interrogation, le candidat est invité à indiquer très brièvement les questions qu'il a abordées lors de la préparation). Il ne sert donc à rien d'aller résoudre des questions faciles de la fin du sujet si l'on n'a pas convenablement traité le début du sujet, pour gagner des points.

Comme pour une épreuve écrite, il est vivement recommandé aux candidats (trop peu le font) de ne pas hésiter à employer dans leurs algorithmes/programmes des fonctions annexes qui allègeront leur code et simplifieront d'une part la présentation et d'autre part l'analyse de complexité. Il est également conseillé d'annoncer à l'oral l'idée générale de l'algorithme avant de se lancer dans une écriture détaillée du code. En particulier il vaut mieux ne pas commencer la présentation de sa solution par l'écriture complète des fonctions auxiliaires. Il peut être utile de prendre des exemples.

Le jury a systématiquement demandé de prouver la correction des algorithmes non triviaux donnés par les candidats. C'est un point qui a souvent posé des difficultés, et tout particulièrement pour les algorithmes récursifs. Par ailleurs, c'est souvent lors de cette étape que le candidat réalise de lui-même que son algorithme est faux : ainsi, nous ne saurions que trop recommander aux candidats d'anticiper la question de la correction pour ainsi détecter lors de la préparation d'éventuelles erreurs. Signalons que donner un algorithme, dont on sait pertinemment qu'il est faux, en espérant que les examinateurs ne s'en rendront pas compte n'est pas une bonne idée : la bonne attitude étant plutôt d'annoncer que l'on n'a pas su trouver une réponse correcte et d'expliquer les idées envisagées et leurs limites.

Le jury a également souvent demandé aux candidats de préciser la complexité de leurs programmes. C'est également un point qui a posé problème aux candidats (et qui, tout comme la capacité à établir la correction des algorithmes, permet de discriminer les candidats), alors que dans certains cas il n'y avait pas de réelles difficultés. Il est ici important de souligner que pour des algorithmes complexes (typiquement récursifs) nous ne demandions pas la complexité exacte mais simplement d'établir des relations de récurrence sans forcément les résoudre (sauf si cela est explicitement un exemple du programme). Par ailleurs les candidats devraient être vigilants sur le fait que lorsqu'un algorithme renvoie un objet de grande taille il est illusoire d'espérer une complexité inférieure à celle de la taille de l'objet retourné.

On a vu également trop souvent des candidats faire des fautes de *typage* en particulier lors de la manipulation de liste de listes.

Un trop grand nombre de candidats ne s'appuient pas sur un exemple pour expliquer un raisonnement complexe ou pour chercher au tableau, la solution à une question non traitée.

La complexité des fonctions de Python est en général bien comprise. Par exemple, la concaténation de deux listes, les slices de Python ou la concaténation de deux chaînes de caractères sont de moins en moins considérées comme des opérations unitaires ce qui est une bonne chose.
Quelques candidats évaluent la complexité d'une fonction en ne regardant que l'imbrication des boucles. Ils négligent ainsi de prendre en compte la complexité de certaines instructions qui n'ont pas un coût élémentaire. A l'inverse, ils considèrent qu'une imbrication de deux boucles *while* est nécessairement de complexité quadratique ce qui les handicape pour analyser des algorithmes non-triviaux.

On déplore que de trop nombreux candidats soient complètement démunis lorsqu'il s'agit de donner une preuve formelle d'une propriété mathématique sur les objets manipulés, ce qui *de facto* les pénalise pour concevoir les algorithmes qui en découlent.

Enfin nous avons pu constater avec satisfaction que le concept de mémoïsation est maitrisé par plusieurs candidats même si parfois il pèche dans l'analyse de la complexité en lien avec les dictionnaires.