Note sur le « sujet zéro » pour l'épreuve d'informatique

Les sujets proposés

Les sujets des épreuves écrites de mathématiques du CAPES sont généralement formés de deux problèmes indépendants ; ce choix permet à la plupart des candidats de trouver sur au moins l'un des deux problèmes une source d'inspiration, ce qu'un problème unique ne permettrait pas aussi bien. Il en sera de même pour l'informatique, et pour les mêmes raisons.

Les deux problèmes conjointement fournis sous le nom de « sujet zéro » sont indépendants ; le sujet au format PDF et son code source LaTeX sont disponibles. Ils sont volontairement assez courts afin d'inciter les préparateurs à les modifier, développer ou combiner avec des sujets de leur cru. On pourra ainsi, par exemple, prolonger l'étude des mots de De Bruijn par leur application au séquençage de l'ADN[1] et celle des L-systèmes avec la problématique de la sélection des L-systèmes les mieux « réussis » (équilibrés, poussant en hauteur, etc.)[2]. On notera que ces deux problèmes sont inspirés de travaux de recherche datant de moins de 40 ans (l'informatique est une science jeune) !

À propos des bibliothèques logicielles

L'usage des fonctions (et classes) issues de modules ou « packages » Python bien établis fait partie des démarches recommandées pour une conception efficace de programmes complexes. Vis-à-vis des épreuves du concours, la connaissance de ces bibliothèques logicielles n'est cependant pas un attendu, et toutes les informations nécessaires seront rappelées dans les annexes des sujets.

Le lien entre mathématiques et informatique

S'agissant d'une épreuve d'un concours de recrutement de professeurs de mathématiques, on ne sera pas surpris d'y trouver des questions ayant un aspect mathématique. C'est ainsi, par exemple, que les questions de complexité débouchent sur la comparaison asymptotique de suites, tandis que celles issues de l'informatique graphique reposent clairement sur la géométrie des vecteurs.

Les compétences attendues et leur évaluation

L'informatique est un domaine qui se prête tout à fait à une évaluation par compétences. Nous pouvons, en cette matière, nous référer aux compétences publiées dans les programmes de CPGE[3] :

Analyser et modéliser	un problème, une situation ;
Imaginer et concevoir	une solution algorithmique modulaire, utilisant des méthodes de programmation et des structures de données appropriées pour le problème étudié ;
Traduire	un algorithme dans un langage de programmation moderne et généraliste ;
Spécifier	rigoureusement les modules ou fonctions ;
Évaluer, contrôler, valider	des algorithmes et des programmes ;
Communiquer	à l'écrit ou à l'oral, une problématique, une solution ou un algorithme, une documentation.

C'est dans cette optique que le questionnement peut revêtir des formes variées ; on pourra ainsi trouver tour à tour :

1 http://www.datagenetics.com/blog/october22013/index.html
http://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_dbg.pdf
http://www.sciencedirect.com/science/article/pii/S0888754310000492
2 http://www.cs.stir.ac.uk/~goc/papers/GAsandL-systems.pdf
3 http://cache.media.education.gouv.fr/file/special_3_ESR/50/5/programme-informatique_252505.pdf

– la recherche d'un algorithme pour résoudre un problème (l'algorithme pouvant être rédigé « en français ») ;

– la preuve de la correction d'un algorithme ;

– l'analyse de la complexité d'un algorithme ;

– l'analyse d'un code de fonction Python fourni, avec pour but de déterminer quelle est son action et quels sont ses résultats ;

– la détection, et possiblement la correction, d'une ou plusieurs erreurs ou faiblesses dans un code de fonction ou de programme exprimé en langage Python ;

– la complétion d'une fonction Python pour obtenir le résultat attendu ;

– l'écriture, en langage Python, d'un code d'une fonction implémentant un algorithme.

L'évaluation des codes écrits en langage Python ne se limite pas à l'aspect « opérationnel » (le code fonctionne, ou ne fonctionne pas). Les correcteurs pourront apprécier un code selon divers facteurs de qualité :

– lisibilité : le code doit être raisonnablement aéré, et commenté quand cela est vraiment utile ;

– documentation : le code d'une fonction doit être accompagné d'une **brève description initiale** du but général de la fonction, ainsi que de la « signature » de la fonction (à savoir, le type des arguments (entiers, flottants, listes etc.) et conditions requises sur ceux-ci, et le type des données renvoyées) ; par ailleurs, **le nom des variables** doit être choisi pour faciliter la compréhension de leur rôle ;

– correction : le code doit pouvoir s'exécuter (quand les entrées sont valides) sans erreur ni boucle infinie, et retourner la valeur attendue ;

– syntaxe : sans tomber dans une obsession des détails, la syntaxe du langage Python (généralement appréciée pour sa concision) impose quelques contraintes comme :

  ▷ une claire distinction entre = et == ;

  ▷ une indentation correcte au moyen de tabulations (qui seront rendues grâce à une barre verticale sur les copies).

  ▷ un caractère deux-points (:) pour délimiter le début et la suite d'une instruction composée (après les instructions conditionnelles, préambules de définitions de fonctions, déclarations de boucles…) ;

  Du fait que les candidats rédigent un code sur papier et non dans un environnement de développement, la correction devra faire preuve d'une certaine indulgence sur ce point.

– fiabilité : les codes susceptibles de provoquer des erreurs à l'exécution (comme les divisions par zéro ou les indices hors limites) seront moins appréciés que les codes robustes[4] ;

– performance et efficacité : au-delà des questionnements spécifiques sur la complexité des algorithmes, un code écrit avec un certain souci d'optimisation sera plus apprécié qu'un code très inefficace ou consommant de la mémoire sans mesure (ce point n'ayant de sens que si le code est déjà considéré comme satisfaisant selon les items précédents, et si les données utilisées sont susceptibles d'être de grande taille).

Un exemple

De ces différents points de vue, considérons la fonction suivante, réalisant la très classique recherche dans un tableau trié :

4 En particulier, le fait que le langage Python soit faiblement typé oblige à effectuer quelques vérifications de domaines de validité.

```
1     def recherche(x,t):
2           if len(t) == 0: return None
3           a = 0
4           b = len(t)-1
5           while a <= b:
6                 c = int((a+b)/2)
7                 if x == t[c]: return c
8                 elif x < t[c]: b = c - 1
9                 else a = c + 1
10          return None
```

Selon le questionnement amenant à écrire le code de cette fonction, on pourra prendre en compte certaines des remarques suivantes (mais pas toutes) :

Lisibilité	Le code est assez lisible, quoique l'écriture des conditionnelles (lignes 7, 8) sur une seule ligne ne soit pas toujours recommandable (ce n'est pas un problème ici).
Documentation	(ligne 1) Il n'y a aucune documentation ; le minimum est ici de préciser le but visé, le domaine de validité de la fonction, ici un entier pour le paramètre `x`, une liste d'entiers triée en croissant pour le paramètre `t` et le type de retour, ici un entier.
Fiabilité	(ligne 6) Le calcul du milieu en passant par une division et une partie entière n'est pas très satisfaisant (d'autant plus que le résultat de cette opération dépend tant de la version du langage Python ; il vaudrait mieux écrire : `c = (a+b)//2`. [5]
Correction	On peut constater que l'algorithme sous-jacent se termine et renvoie l'indice d'un élément du tableau `t` égal à `x`, si `x` est dans le tableau, et `None` sinon[6]. Lorsque la valeur `x` est présente plusieurs fois dans le tableau, la valeur de l'indice renvoyé est ambiguë.
Syntaxe	(ligne 9) Le code est incorrect en un endroit (il manque un : après le `else`).
Performance et efficacité	La recherche dichotomique est un algorithme optimal (cela peut constituer le thème d'une question à part). Un algorithme de balayage du tableau (dans l'ordre) serait considéré comme peu efficace en comparaison[7].

5 Très accessoirement, le comportement de la fonction `recherche` n'est guère informatif quand `x` n'est pas dans le tableau ; l'énoncé pourrait éventuellement demander une variante dans laquelle la valeur retournée serait l'indice du plus grand élément du tableau inférieur ou égal à `x` (deux petites modifications à apporter).

6 La justification de tout cela pourrait éventuellement être demandée dans l'énoncé.

7 Tout dépend cependant de la taille du tableau : si cette taille est de quelques dizaines d'éléments la discussion n'a pas d'intérêt.