

BACCALAURÉAT GÉNÉRAL

CORRECTION DE L'ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : 3 heures 30

Le sujet est composé de trois exercices indépendants.

Exercice 1		6 points		
Questions	Contenu et notions	Capacités exigibles / Niveau	Barème	Éléments de réponses et commentaires
1	Analyser un problème	Analyser et modéliser un problème / N1	0,5	L'élément absolument majoritaire d'une liste de taille 10 est d'effectif 6, 7, 8, 9 ou 10.
2	Programmation Python	Parcours séquentiel d'un tableau / N1	0,5	<pre> 1 def effectif(elt, lst): 2 somme = 0 3 for e in lst: 4 if e == elt: 5 somme = somme + 1 6 return somme </pre>
3	Analyser un programme	Analyser et modéliser un problème / N1	0,25	L'appel à <code>effectif(1, [1, 4, 1, 6, 1, 7, 2, 1, 1])</code> fait 9 comparaisons.
4	Programmation Python	Parcours séquentiel d'un tableau / N2	0,5	<pre> 1 def majo_abs1(lst): 2 n = len(lst) 3 for e in lst: 4 eff = effectif(e, lst) 5 if eff > n/2: 6 return e 7 return None </pre>
5	Analyser un programme	Analyser et modéliser un problème / N2	0,5	l'appel à <code>majo_abs1([1, 4, 1, 6, 1, 7, 2, 1, 1])</code> fait $9 \times 9 = 81$ comparaisons.
6	Programmation Python	Parcours séquentiel d'un tableau, d'un dictionnaire / N1	0,5	<pre> 1 def eff_dico(lst): 2 dico_sortie = {} 3 for e in lst: 4 if e in dico_sortie: 5 dico_sortie[e] = dico_sortie[e] + 1 6 else: 7 dico_sortie[e] = 1 8 return dico_sortie </pre>

Exercice 1		6 points		
7	Programmation Python	Parcours séquentiel d'un tableau, d'un dictionnaire / N2	1	<pre>def majo_abs2(lst): dico = eff_dico(lst) elt_max = "" eff_max = -float("inf") for e in dico: if dico[e] > eff_max: eff_max = dico[e] elt_max = e if eff_max > len(lst)/2: return elt_max else: return None</pre>
8	Déterminer un cas de base en récursivité	Ecrire un algorithme utilisant la méthode "diviser pour régner" / N1	0,25	Si $n = 1$, alors l'élément absolument majoritaire de <code>lst</code> est <code>lst[0]</code> .
9	Concevoir un algorithme récursif	Ecrire un algorithme utilisant la méthode "diviser pour régner" / N3	0,5	Si <code>lst1</code> est de taille $n1$, alors ses éléments sont d'effectif au plus $n1/2$. Si <code>lst2</code> est de taille $n2$, alors ses éléments sont d'effectif au plus $n2/2$. Donc les effectifs des éléments de <code>lst</code> sont au plus $n1/2 + n2/2 = n/2$. Ainsi <code>lst</code> n'admet pas d'élément absolument majoritaire.
10	Concevoir un algorithme récursif	Ecrire un algorithme utilisant la méthode "diviser pour régner" / N3	0,5	Si <code>lst1</code> admet un élément absolument majoritaire <code>maj1</code> , alors <code>maj1</code> est un candidat pour être l'élément absolument majoritaire de <code>lst</code> . il suffit de déterminer l'effectif de <code>maj1</code> dans <code>lst</code> .

Exercice 1		6 points	
11	Programmation Python	Traduire un algorithme dans un langage de programmation / N3	1
			<pre> 1 def majo_abs3(lst): 2 n = len(lst) 3 if n == 1: 4 return lst[0] 5 else: 6 lst_g = lst[:n//2] 7 lst_d = lst[n//2:] 8 maj_g = majo_abs3(lst_g) 9 maj_d = majo_abs3(lst_d) 10 if maj_g is not None: 11 eff = effectif(maj_g, lst_g) + effectif(maj_g, lst_d) 12 if eff > n/2: 13 return maj_g 14 if maj_d is not None: 15 eff = effectif(maj_d, lst_g) + effectif(maj_d, lst_d) 16 if eff > n/2: 17 return maj_d </pre>

Exercice 2 6 points				
<i>Questions</i>	<i>Contenu et notions</i>	<i>Capacités exigibles / Niveau</i>	<i>Barème</i>	<i>Éléments de réponses et commentaires</i>
1	Analyser un problème	Analyser et modéliser un problème / N1	0,5	<code>[2*(i+1)-3 for i in range(3, 10)]</code> n'est pas bien parenthésée car <code>**2*(i+1)-3**</code> n'est pas bien parenthésée.
2	Programmation Python	Parcours séquentiel d'une chaîne de caractères / N1	0,5	code : <pre>def compte_ouvrante(txt): par = ["(", "{", "["] eff = 0 for car in txt: if car in par: eff = eff + 1 return eff</pre>
3	Programmation Python	Parcours séquentiel d'une chaîne de caractères / N1	0,5	code : <pre>def compte_fermante(txt): par = [")", "}", "]"] eff = 0 for car in txt: if car in par: eff = eff + 1 return eff</pre>
4	Programmation Python	Parcours séquentiel d'une chaîne de caractères / N1	0,5	code : <pre>def bon_compte(txt): return compte_ouvrante(txt) == compte_fermante(txt)</pre>
5	Analyser un programme	Analyser et modéliser un problème / N2	0,5	<code>bon_compte("tab([1])")</code> renvoie True mais <code>"tab([1])"</code> n'est pas bien parenthésée.

Exercice 2		6 points		
6	Structure de Piles et POO	Spécifier une structure de données par son interface et POO / N2	1	code : <pre> 1 class Pile: 2 def __init__(self): 3 self.contenu = [] 4 5 def est_vide(self): 6 return len(self.contenu) == 0 7 8 def empiler(self, elt): 9 self.contenu.append(elt) 10 11 def depiler(self): 12 if self.est_vide(): 13 return "La pile est vide." 14 return self.contenu.pop()</pre>
7	Analyser un algorithme	Analyser un algorithme / N2	1	Il y aura au plus $n + n/2$ comparaisons ($n/2$ pour les parenthèses fermantes). A la place de $n/2$, on peut accepter “nb_fermantes” (ou équivalent).
8	Programmation Python	Traduire un algorithme dans un langage de programmation / N3	1,5	code : <pre> def est_bien_parenthesee(expr): parentheses = {'{' : '}', '[' : ']', '(' : ')'} pile = Pile() for e in expr: if e in parentheses.keys(): pile.empiler(e) if e in parentheses.values(): if pile.est_vide(): return False else: f = pile.depiler() if parentheses[f] != e: return False return pile.est_vide()</pre>

Exercice 3		8 points		
Questions	Contenu et notions	Capacités exigibles / Niveau	Barème	Éléments de réponses et commentaires
1	base de données	N1	0.5 (0.25 titre et 0.25 album)	Non car il peut exister deux chansons qui ont le même titre. Par exemple la chanson Showbiz.
2	base de données	N1	0.5 (0.25 s'il manque 1 ligne, 0 s'il manque 2 lignes ou plus)	Voir le tableau 1.
3	base de données	N1	0.5	SELECT titre FROM Chanson WHERE album = 'Showbiz' ORDER BY titre;
4	base de données	N1	0.5	INSERT INTO Chanson VALUES (10, 'Megalomania', 'Hullabaloo', 'Muse');
5	base de données	N2	0.5	UPDATE Chanson SET titre = 'Welcome to the Jungle' WHERE id = 7;
6	base de données	N1	0.25	éviter la duplication de données.
7	base de données	N1	0.25	id_album est une clé étrangère pour la table Chanson.
8	base de données	N1	0.75 (0.25 par relation)	Chanson (<u>id INT</u> (clé primaire soulignée), titre TEXT, #id_album INT (clé étrangère)) Album (<u>id INT</u> (clé primaire soulignée), titre TEXT, année INT, #id_groupe INT (clé étrangère)) Groupe (<u>id INT</u> (clé primaire soulignée), nom TEXT)
9	base de données	N2	0.75	SELECT a.titre FROM Album AS a JOIN Chanson AS c ON c.id_album = a.id WHERE c.titre = 'Showbiz';

Exercice 3		8 points		
10	base de données	N2	0.75	<pre> SELECT c.titre, a.titre FROM Chanson AS c JOIN album AS a ON a.id=c.id_album JOIN Groupe AS g ON g.id=a.id_groupe WHERE g.nom='Muse'; </pre>
11	base de données	N2	0.25	Le nombre d'albums du groupe Muse.
12	Programmation Python	N1	0.5 (0.25 par réponse)	<pre> 1 assert ordre_lex('', 'a') == True 2 assert ordre_lex('b', 'a') == False 3 assert ordre_lex('aaa', 'aaba') == True </pre>
13	Programmation Python ; Récursivité	N2	1 (0.25 True, 0.25 False, 0.5 appel récursif)	<pre> 1 def ordre_lex(mot1, mot2): 2 if mot1 == '': 3 return True 4 elif mot2 == '': 5 return False 6 else: 7 c1 = mot1[0] 8 c2 = mot2[0] 9 if c1 < c2: 10 return True 11 elif c1 > c2: 12 return False 13 else: 14 return ordre_lex(mot1[1:], mot2[1:]) </pre>

Exercice 3	8 points				
14	Programmation Python	N3	1	<pre> 1 def ordre_lex(mot1, mot2): 2 if mot1 == '': 3 return True 4 elif mot2 == '': 5 return False 6 else: 7 i = 0 8 while i < len(mot1) and i < len(mot2): 9 c1 = mot1[i] 10 c2 = mot2[i] 11 if c1 < c2: 12 return True 13 elif c1 > c2: 14 return False 15 else: 16 i = i + 1 17 return len(mot1) <= len(mot2) </pre>	

Tableau 1.

	titre	album
	Sunburn	Showbiz
	Muscle Museum	Showbiz
	Showbiz	Showbiz
	New Born	Origin of Symmetry
	Sing for Absolution	Absolution
	Hysteria	Absolution
	Muscle Museum	Hullabaloo
	Showbiz	Hullabaloo