Programmation fonctionnelle

Examen du 14 janvier 2022

L2 math-info / L3 info – Portail Sciences – Université Côte d'Azur

Durée : 2 heures.

— Aucun document ni aucune machine ne sont autorisés. Un mémo est donné en fin de sujet.
— Les téléphones doivent être rangés.
— Les réponses sont à reporter sur les feuilles de réponse en fin de sujet. Ces feuilles sont à dégrafer du sujet et à rendre en fin d'épreuve sans agrafe.
— Les réponses doivent être écrites lisiblement. Le correcteur blanc et l'effaceur sont autorisés, ils peuvent être utilisés pour décocher une case cochée par erreur, mais dans ce cas, n'essayez pas de redessiner la case.
— Les questions sont notées de manière indépendante : une question peut être sautée, et une fonction `f` demandée à une question peut être utilisée pour définir une fonction `g` dans une question ultérieure même si la solution pour `f` n'a pas été trouvée. On ne peut pas utiliser la fonction d'une question pour répondre à une question précédente (par exemple, on ne peut pas utiliser la fonction de la question 7 pour répondre à la question 2).
— Il est recommandé de traiter les exercices et les questions dans l'ordre, sans bloquer sur une question. Certaines questions portent une indication de difficulté, forcément subjective.
— Le barème est donné à titre indicatif et pourra être revu pour améliorer la moyenne globale de la promo. Le barème prévisionnel comporte 30 points. Il suffira de valider 20 points pour obtenir 20/20.

Numération ternaire balancée (12 points)

La numération ternaire balancée [1] est un système de numération en base 3 utilisant trois chiffres, parfois appelés *trits*, que l'on notera 0, 1, et Ɪ, et valant respectivement les nombres 0, 1, et -1. Par exemple, le nombre moins trente trois, que l'on note $(-33)_{10}$ en numération décimale classique, s'écrit $(\text{Ɪ}\text{Ɪ}10)_{3b}$ en numération ternaire balancée, car

$$(-33)_{10} = (-27)_{10} + (-9)_{10} + (3)_{10} = (\text{Ɪ})_{3b} \times 3^3 + (\text{Ɪ})_{3b} \times 3^2 + (1)_{3b} \times 3^1 + (0)_{3b} \times 3^0.$$

La numération ternaire balancée a été expérimentée dans les années soixante sur quelques ordinateurs soviétiques, les « ternacs » [2].

Question 1 (facile) On pose `type trit = Z | U | T` le type des chiffres en numération ternaire balancée, avec `Z` pour 0 (**z**éro), `U` pour 1 (**u**n), et `T` pour Ɪ (**t**ourné). Définissez la fonction `trit_to_int: trit -> int` qui à un trit associe sa valeur. Par exemple, `trit_to_int T` renvoie `-1`.

Question 2 (facile) On pose `type be3b_number = trit list`. Un objet de ce type représente une écriture en style *gros-boutiste* (en anglais *big endian*) d'un nombre en base ternaire balancée. Par exemple, `[Z; U; T]` représente le nombre $(01\text{Ɪ})_{3b} = (2)_{10}$. Cette écriture du nombre $(2)_{10}$ n'est pas normalisée car elle commence par un zéro. Définissez la fonction `be3b_normalize: be3b_number -> be3b_number` qui renvoie la version normalisée d'un nombre : on retire tous les zéros inutiles en tête de liste, sauf le dernier si le nombre représenté vaut 0. Par exemple, `be3b_normalize [Z;U;T]` renvoie `[U;T]`, et `be3b_normalize [Z;Z;Z]` renvoie `[Z]`.

Question 3 (facile) Recopiez et complétez la table d'addition en numération ternaire balancée, puis définissez la fonction `trit_add2: trit -> trit -> trit * trit` qui à deux trits t_1 et t_2 associe le couple de trits (c, t) tel que ct soit l'écriture gros-boutiste de $t_1 + t_2$ sur deux trits (la « retenue » c peut être nulle). Par exemple `trit_add2 U U` renvoie `(U, T)`, tandis que `trit_add2 T U` renvoie `(Z, Z)` (voir table ci-dessous). Organisez votre code au mieux pour regrouper le maximum de cas.

+	0	1	Ɪ
0			
1		1Ɪ	00
Ɪ		00	

Question 4 (facile) Définissez *sans utiliser les mot-clés* `rec` *et* `while`, mais avec une fonction d'ordre supérieur sur les listes, la fonction `be3b_uminus: be3b_number -> be3b_number` qui à l'écriture de l'entier n associe l'écriture de l'entier $-n$. Indication : il faut remplacer chaque chiffre par son opposé. Par exemple, `be3b_uminus [Z; U; T]` renvoie `[Z; T; U]`. Une solution qui utilise le mot-clé `rec` donnera quelques points.

1. voir https ://en.wikipedia.org/wiki/Balanced_ternary
2. notamment le Setun mis au point par l'équipe du mathématicien Sergueï Sobolev et utilisé pour exécuter l'assistant éducatif Nastavnik.

Pour votre examen, imprimez de préférence les documents compilés à l'aide de auto-multiple-choice.

Question 5 Définissez *sans utiliser les mot-clés* `rec` *et* `while`, mais avec une fonction d'ordre supérieur sur les listes, la fonction `be3b_even: be3b_number -> bool` qui renvoie `true` si l'écriture dénote un entier pair. Indication : il faut vérifier si la somme des chiffres est paire. Par exemple, `be3b_even [Z; U; T]` renvoie `true` et `be3b_even [T; U; T]` renvoie `false` . Une solution qui utilise le mot-clé `rec` donnera quelques points.

Question 6 Définissez *sans utiliser les mot-clés* `rec` *et* `while`, mais en utilisant une fonction d'ordre supérieur sur les listes, la fonction `be3b_to_int: be3b_number -> int` qui renvoie l'entier Caml associé à une écriture gros-boutiste d'un nombre en base ternaire balancée. Par exemple, `be3b_to_int [Z; U; T]` renvoie l'entier Caml de valeur $(01\bar{1})_{3b} = (2)_{10}$. La fonction n'a besoin d'être correcte que pour les écritures dont la valeur est représentable par un entier Caml (autrement dit, faites vos calculs en ignorant les dépassements de capacité). Une solution avec le mot-clé `rec` rapportera des points.

Question 7 (facile) On pose `type le3b_number = trit list`. Un objet de ce type représente une écriture en style *petit-boutiste* (en anglais *little endian*) d'un nombre en base ternaire balancée : l'ordre des chiffres dans la liste est l'ordre de lecture de droite à gauche. Par exemple, `[Z; U; T]` représente le nombre $(\bar{1}10)_{3b} = (-9+3)_{10} = (-6)_{10}$. L'écriture `[Z; U; T]` est ici normalisée car la liste ne termine pas par un zéro. Définissez la fonction `le3b_normalize: le3b_number -> le3b_number` qui renvoie la version normalisée d'un nombre : on retire tous les zéros inutiles en fin de liste, sauf un si le nombre représenté vaut 0.

Question 8 Définissez la fonction `le3b_sign: le3b_number -> trit` qui renvoie le signe du nombre représenté : la fonction renvoie `T` si le nombre est strictement négatif, `Z` si le nombre est nul, et `U` si le nombre est strictement positif. Par exemple, `le3b_sign [Z;U;T]` renvoie `T`, `le3b_sign [T;U;T;U]` renvoie `U`, et `le3b_sign [T;U;Z;Z]` renvoie `U`. Indication : il suffit de renvoyer le dernier chiffre non nul, s'il existe, sinon `Z`.

Question 9 (facile) Définissez la fonction `le3b_to_int: le3b_number -> int` qui renvoie l'entier Caml associé à une écriture petit-boutiste d'un nombre en base ternaire balancée. Par exemple, `le3b_to_int [Z; U; T]` renvoie l'entier Caml de valeur $(-6)_{10}$. La fonction n'a besoin d'être correcte que pour les écritures dont la valeur est représentable par un entier Caml.

Question 10 En réutilisant la fonction `trit_add2` précédente, définissez la fonction `trit_add3: trit -> trit -> trit -> trit * trit` qui à trois trits t_1, t_2, t_3 associe le couple (c, t) tel que ct soit l'écriture gros-boutiste sur deux trits de $t_1 + t_2 + t_3$ (c correspond à la « retenue »). Par exemple, `trit_add3 T T T` renvoie `(T,Z)`.

Question 11 (moyen/difficile) Définissez la fonction `le3b_add: le3b_number -> le3b_number -> le3b_number` qui à deux écritures petit-boutistes dénotant des entiers n et m associe l'écriture petit-boutiste de $n+m$. Indication : on effectuera l'algorithme d'addition « classique », en partant des chiffres les moins significatifs (en début de liste), et en propageant la retenue. Par exemple, `le3b_add [Z; U; T] [T; U; T; U]` renvoie `[T; T; T; U]`.

Question 12 (moyen/difficile) Définissez la fonction `int_to_le3b: int -> le3b_number` qui à un entier Caml associe l'écriture petit-boutiste normalisée de cet entier en base ternaire balancée.

Indication : on peut passer de la base 3 à chiffres positifs au ternaire balancé en ajoutant $1111\ldots$ avec retenue, puis en soustrayant $1111\ldots$ chiffre à chiffre, sans retenue. Par exemple, $(021)_3 + (111)_3 = (202)_3$, et quand on soustrait chiffre à chiffre 111 à 202, on obtient $1\bar{1}1$, et on a bien $(021)_3 = (1\bar{1}1)_{3b} = (7)_{10}$.

Nombres réels effectivement approximables (6 points)

Dans un système de numération complet de base β, on peut écrire tout nombre réel d'au moins une façon sous la forme $\beta^e \times 0, b_0 b_1 \ldots$ où l'*exposant* e est un entier signé et la *mantice* $b_0 b_1 \ldots$ est une suite infinie de chiffres. Par exemple, π s'écrit dans le système décimal $10^1 \times 0,31415\ldots$. Un nombre réel est effectivement approximable (dans un système de numération positionnelle donné) s'il admet une écriture telle que e et la fonction $i \mapsto b_i$ sont calculables[3]. Dans cet exercice on va s'intéresser à définir l'addition de deux réels effectivement approximables. Ceci n'est pas possible dans le système de numération binaire (et plus généralement dans tout système positionnel à chiffres positifs) à cause de l'effet domino d'une propagation de retenue ; en effet, si on veut calculer la somme

$$\begin{array}{rr} & 0,101010\ldots \\ + & 0,010101\ldots \\ \hline & ?,\ldots \end{array}$$

3. La plupart des nombres réels que vous avez croisé en math ou en physique sont effectivement approximables : les nombres algébriques, la constante de Neper e, le nombre π, la constante d'Euler-Mascheroni γ, et plus généralement tout nombre qui s'exprime à l'aide de séries ou de produits infinis dénombrables est effectivement approximable dans tout système de numération raisonnable.

Pour votre examen, imprimez de préférence les documents compilés à l'aide de auto-multiple-choice.

on a besoin d'une « boule de cristal » pour décider si le bit avant la virgule dans le résultat vaut 0 ou 1 :

$$\begin{array}{rl} & 0,101010\ldots \text{ (à l'infini)} \\ + & 0,010101\ldots \text{ (à l'infini)} \\ \hline & 0,111111\ldots \\ \text{ou} & 1,000000\ldots \end{array} \qquad \begin{array}{rl} & 0,101010\ldots 11\ldots \\ + & 0,010101\ldots 01\ldots \\ \hline & 1,000000\ldots 1? \end{array} \qquad \begin{array}{rl} & 0,101010\ldots 00\ldots \\ + & 0,010101\ldots 01\ldots \\ \hline & 0,111111\ldots ??\ldots \end{array}$$

Pour pouvoir additionner deux réels effectivement approximables, on va utiliser un système de numération redondant à chiffres signés. Ces systèmes de numération ont été découverts par Algirdas Antanas Avižienis au siècle dernier, au début des années soixante, et ont la propriété de permettre de paralléliser les additions (peut-être en avez-vous entendu parlé en cours d'architecture avec M. Touati ?). Nous allons nous intéresser à l'addition dans le système d'Avižienis binaire [4], autrement dit en base 2 (c'est donc un peu différent du système étudié à l'exercice précédent), et nous suivrons la présentation simplifiée qu'en a donné Martín Escardó en 2015 [5], basée sur l'utilisation de deux jeux de chiffres signés différents : les *trits* ($\bar{1}, 0, 1$ vus à l'exercice précédent) et les *quints* ($\bar{2}, \bar{1}, 0, 1, 2$).

On pose les définitions suivantes

```
type 'a item = Item of 'a * 'a flot
and 'a flot = 'a item Lazy.t
type trit = int (* -1, 0, 1 *)
type quint = int (* -2, -1, 0, 1, 2 *)
type aviz_trit_real = trit flot
type aviz_quint_real = quint flot
let rec forever n = lazy(Item(n, forever n))
let rec aviz_to_float n x =
   if n=0 then 0.0
   else let Item(b0, x') = Lazy.force x
        in 0.5 *. ((float b0) +. (aviz_to_float (n-1) x'))
```

Question 13 Que vaut `aviz_to_float n (forever 1)` en ignorant les erreurs d'arrondi sur les flottants ?

Question 14 Définissez la fonction `aviz_add: aviz_trit_real -> aviz_trit_real -> aviz_quint_real` qui effectue la somme de deux réels d'Avižienis chiffre à chiffre, sans retenue, en utilisant pour écrire le résultat les cinq chiffres $\bar{2}, \bar{1}, 0, 1, 2$. Par exemple, la somme de $0,1\bar{1}0\ldots$ et $0,\bar{1}\bar{1}1\ldots$ vaut $0,0\bar{2}\bar{1}\ldots$.

Question 15 Définissez la fonction `aviz_div2: aviz_quint_real -> aviz_trit_real` qui à l'écriture d'Avižienis d'un nombre réel $x \in [-2, 2]$ dans le jeu de chiffres $\bar{2}, \bar{1}, 0, 1, 2$ associe une écriture du nombre réel $\frac{x}{2} \in [-1, 1]$ avec le jeu de chiffres $\bar{1}, 0, 1$. Par exemple, on aura

$$\begin{array}{rl} x = & 0,\ 2\ 0\ \bar{2}\ 10\ 1\bar{1}0\ 1\bar{2}\ \ldots \\ \frac{x}{2} = & 0,\ 1\ 0\ \bar{1}\ 01\ 001\ 00\ \ldots \end{array}$$

Indication : si les listes Caml étaient paresseuses comme en Haskel, on pourrait écrire la fonction demandée de la façon suivante. [a]

```
let rec aviv_div2 x = match x with
| 0 :: x' -> 0 :: aviz_div2 x'
| 2 :: x' -> 1 :: aviz_div2 x'
| -2 :: x' -> -1 :: aviz_div2 x'
| a :: b :: x' ->
   let d = 2*a+b in
   if d < -2 then -1 :: aviz_div2 (d+4 :: x')
   else if d > 2 then 1 :: aviz_div2 (d-4 :: x')
   else 0 :: aviz_div2 (d :: x')
```

a. Quelques explications sur ce que fait ce code : on travaille de gauche à droite (les chiffres les plus significatifs en premier). Si on doit diviser par deux un $\bar{2}$, un 0, ou un 2, on sait faire. Pour diviser un 1, on a deux possibilités : on peut le remplacer par un 0 et créer une « retenue à droite » égale à 1, ou bien on peut garder le 1 et créer une retenue à droite égale à $\bar{1}$. Pour choisir entre les deux, on regarde quel est le chiffre à droite, ceci afin d'éviter que la retenue à droite en s'additionnant au chiffre à droite divisé par deux crée une retenue à gauche. On procède de manière similaire pour diviser un $\bar{1}$ par deux.

Question 16 (facile) Déduire des deux questions précédentes la fonction `aviz_mid: aviz_trit_real -> aviz_trit_real -> aviz_trit_real` qui à deux réels x, y effectivement approximables en base binaire d'Avižienis associe leur moyenne $\frac{x+y}{2}$.

4. dû non pas à Avižienis, mais à Chow et Robertson, en 1978, cf https ://fr.wikipedia.org/wiki/Système_de_numération_d%27Avizienis.
5. cf https ://www.cs.bham.ac.uk/~mhe/papers/fun2011.lhs

Récursivité terminale, fichiers, exceptions, et typage (6 points)

On considère le code suivant

```
let f ic =
   let rec iter res =
     try input_line ic; iter (res+1)
     with End_of_file -> res
   in iter 0
```

Question 17 Que vaut `f (open_in "toto.txt")` si l'utilisateur a les droits en lecture sur le fichier `toto.txt` ? Et s'il n'a pas les droits ? Expliquez comment procède la fonction pour calculer le résultat.

Question 18 Lorsque le fichier `toto.txt` est volumineux, on observe que cette fonction lève l'exception `Stack_overflow`. Rappelez ce qu'est une fonction récursive terminale, en donnant un exemple, et expliquez pourquoi la fonction `f` ci-dessus n'est pas récursive terminale.

Question 19 Réécrivez la fonction `f` de manière récursive terminale (sans while).

Question 20 On considère une nouvelle version de la fonction `f` précédente.

```
let f2 ic =
   let res = ref 0 in
   try
     while true do
       input_line ic;
       res := !res + 1
     done
   with End_of_file -> !res
```

L'inférence de type affiche le message d'erreur suivant :

```
[...]
with End_of_file -> !res;;
Error: This expression has type int but an expression was expected of type unit
```

Corrigez le code de la fonction `f2` pour ne plus avoir de message d'erreur de typage.

Tableaux persistants arborescents (6 points)

Dans cet exercice on se propose de définir des modules de manipulation de tableaux persistants, comportant les fonctions suivantes

- `make n v` renvoie un tableau de `n` cases initialisées à `v`
- `get tab i` renvoie la valeur de la case d'indice `i` du tableau persistant `tab`
- `let tab2 = set tab1 i v in ...` a pour effet de stocker dans la variable `tab2` une copie du tableau `tab1` dans laquelle le contenu de la case d'indice `i` a été remplacé par la valeur `v`, tandis que le tableau `tab1` reste inchangé (« nos tableaux sont persistants »).
- `len tab` renvoie la longueur du tableau

Question 21 (facile) Donnez la signature `module type PERSISTANT_ARRAY = sig ... end` d'un module de manipulation de tableaux persistants offrant les mêmes fonctionnalités que la signature (non persistante) suivante. Indication : il y a juste une ligne à changer ! Sur la feuille de réponse, écrivez cette ligne.

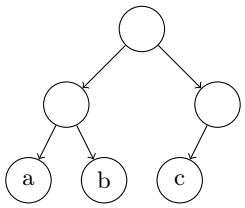
```
module type MUTABLE_ARRAY = sig
   type 'a t
   exception Out_of_bound
   val make : int -> 'a -> 'a t
   val get : 'a t -> int -> 'a
   val set : 'a t -> int -> 'a -> unit
   val len : 'a t -> int
end
```

Pour votre examen, imprimez de préférence les documents compilés à l'aide de auto-multiple-choice.

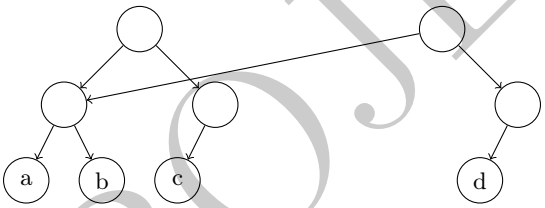
Question 22 (facile) Donnez une implémentation de votre signature `PERSISTANT_ARRAY` basée sur les tableaux de Caml, et précisez la complexité en temps (constant, logarithmique, ou linéaire) de chaque opération. Indication : votre code ressemblera à ceci.

```
module Persistant_Array_Caml : PERSISTANT_ARRAY = struct
   type 'a t = 'a array
   exception Out_of_bound
   let make size default = ...
   ...
end
```

On se propose d'implémenter un tableau persistant par un arbre binaire dont les feuilles représentent les cases. Par exemple, pour représenter le tableau `tab1` que l'on écrirait `[|'a'; 'b'; 'c'|]` avec un `char array` Caml, on aura l'arbre binaire suivant.



Pour implémenter l'opération persistante `let tab2 = set tab1 i v in ...`, on va créer une copie de tous les noeuds intermédiaires sur le chemin de la racine au noeud qui correspond à la case d'indice `i`, et partager les sous-arbres communs avec l'arbre qui code `tab1`. Par exemple, `set tab1 2 'd'` créera, pour représenter `tab2`, l'arbre dont la racine est à droite dans la figure ci-dessous.



Question 23 (moyen/difficile) Donnez une implémentation de votre signature `PERSISTANT_ARRAY` basée sur cette idée. Il y a plusieurs solutions possibles, choisissez celle qui vous semble la plus simple. On demande seulement que `get` et `set` soient en temps logarithmique en la taille du tableau, et `len` en temps constant.

Mémo sur les listes et les couples

— `fst c` : renvoie le premier élément du couple c
exemple :`fst (1,2)` renvoie `1`
— `snd c` : renvoie le deuxième élément du couple c
exemple : `snd (1,2)` renvoie `2`
— `List.rev l` : renvoie l'image mirroir de l
exemple : `List.rev [1;2;3]` renvoie `[3; 2; 1]`
— `List.mem x l` : renvoie `true` si x apparait dans l
exemple : `List.mem 1 [1;2;3]` renvoie `true`
— `List.filter f l` : renvoie la liste des éléments `x` de `l` pour lesquels `f x = true`
exemple : `List.filter (fun x -> x>0) [-1;2;0]` renvoie `[2]`
— `List.map f l` : renvoie la liste des images de f
exemple : `List.map (fun x -> x+2) [1;2;3]` renvoie `[3;4;5]`
— `List.flatten l` : renvoie la concaténation des listes de l
exemple : `List.flatten [[]; [0]; [1;2;3]]` renvoie `[0; 1; 2; 3]`
— `List.fold_left f a0 [a1;a2;...;an]` : renvoie `(f ... (f a0 a1)... an)`
exemple : `List.fold_left (+) 0 [1;2;3;4]` renvoie `10`

Mémo sur les tableaux

— `Array.make n x` : renvoie un tableau de taille `n` contenant `x` dans chaque case
— `Array.length a` : renvoie la longueur du tableau `a`
— `a.(i) <- v` : remplace par `v` le contenu de la case d'indice `i` du tableau `a`
— `Array.copy a` : renvoie une copie du tableau a

Mémo sur les arbres

```
type bintree = Leaf | Node of bintree * bintree
let rec nb_nodes t = match t with
| Leaf -> 0
| Node(l, r) -> 1 + nb_nodes l + nb_nodes r
```

☐0	☐0	☐0	☐0	☐0	☐0	☐0	☐0
☐1	☐1	☐1	☐1	☐1	☐1	☐1	☐1
☐2	☐2	☐2	☐2	☐2	☐2	☐2	☐2
☐3	☐3	☐3	☐3	☐3	☐3	☐3	☐3
☐4	☐4	☐4	☐4	☐4	☐4	☐4	☐4
☐5	☐5	☐5	☐5	☐5	☐5	☐5	☐5
☐6	☐6	☐6	☐6	☐6	☐6	☐6	☐6
☐7	☐7	☐7	☐7	☐7	☐7	☐7	☐7
☐8	☐8	☐8	☐8	☐8	☐8	☐8	☐8
☐9	☐9	☐9	☐9	☐9	☐9	☐9	☐9

← Codez ci-contre votre numéro d'étudiant : cochez dans la première colonne le premier chiffre (a priori un 2), puis dans la deuxième colonne le deuxième chiffre, etc

Écrivez explicitement votre numéro d'étudiant (par précaution) : .....

Les réponses aux questions sont à donner exclusivement sur les feuilles qui suivent : les réponses données sur les feuilles précédentes ne seront pas prises en compte.

Réponse à la question 1 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

Réponse à la question 2 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

Réponse à la question 3 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

Réponse à la question 4 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

Réponse à la question 5 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

Réponse à la question 6 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

Réponse à la question 7 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

Réponse à la question 8 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

Réponse à la question 9 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

Réponse à la question 10 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

Réponse à la question 11 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 12 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

Réponse à la question 13 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

Réponse à la question 14 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

Réponse à la question 15 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 16 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

Réponse à la question 17 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

Réponse à la question 18 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 19 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 20 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 21 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

Réponse à la question 22 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Réponse à la question 23 : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Cases réservées à la correction*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....