1ère Spé. NSI Gr. 2 **Contrôle #2, 18/12/2023**
Lycée Fustel de Coulanges **Durée: 1 Heure**

Ce contrôle comporte 7 questions; il sera noté sur 20 points. Les réponses sont à porter sur une copie comportant votre nom. Il n'est pas nécessaire de répondre aux questions dans l'ordre — commencez par celles où vous vous sentez le plus à l'aise *(mais ne tentez les questions bonus qu'après avoir fini le reste!!)*. Les calculatrices ne sont pas autorisées.

---

1. (1 ½ points) *Questions à choix multiples* (aucune justification de la réponse n'est nécessaire; plusieurs réponses sont possibles):
   (a) L'ordre correct, de la plus petite quantité à la plus grande, est:
   - ◯ **A:** 1 Mo (mégaoctet), 1 bit, 1 Go (gigaoctet), 1 To (téraoctet), 1 ko (kilooctet)
   - √ **B: 1 bit, 1 ko, 1 Mo, 1 Go, 1 To**
   - ◯ **C:** 1 bit, 1 ko, 1 Go, 1 To, 1 Mo
   - ◯ **D:** 1 bit, 1 ko, 1 Mo, 1 To, 1 Go

   (b) Sélectionnez, parmi les affirmations ci-dessous, celle(s) qui est (ou sont) vraie(s):
   - ◯ **A:** Un nombre réel encodé au moyen de la norme IEEE 754 ne peut jamais être une valeur exacte – c'est toujours une valeur approchée.
   - √ **B: Il est dangereux de faire des comparaisons entre nombres flottants car du fait de leur encodage cela peut générer des erreurs.**
   - √ **C: L'encodage des nombres réels au moyen de la norme IEEE 754 s'appuie sur les mêmes principes que la notation scientifique (sauf que l'on utilise des puissances de 2 au lieu de puissances de 10).**

   (c) Sélectionnez, parmi les affirmations ci-dessous, celle(s) qui est (ou sont) vraie(s):
   - √ **A: Un texte codé en ASCII est simplement une suite d'octets correspondant au codage de chacun des caractères du texte.**
   - √ **B: Un texte codé en ASCII ne peut pas contenir de lettres comportant des accents (comme "é" par exemple).**
   - √ **C: L'encodage ASCII est moins complet que l'encodage UTF-8.**
   - ◯ **D:** Un caractère codé en ASCII l'est sur 64 bits.

2. *Addition de nombres binaires* (posez bien votre addition – un simple résultat ne sera pas accepté.)
   (a) (1 point) Effectuez l'addition suivante: 10001011 + 10001110
   (b) (1 point) Est-ce-que cette addition peut être effectuée sans erreur si les entiers sont codés sur 8 bits? Justifiez votre réponse.

Solution:

$$\begin{array}{r} 1\,0\,0\,0\,1\,0\,1\,1 \\ +\quad 1\,0\,0\,0\,1\,1\,1\,0 \\ \hline 1\,0\,0\,0\,1\,1\,0\,0\,1 \end{array}$$

La dernière retenue fait que l'on va devoir utiliser un bit de plus pour encoder le résultat de cette addition – et que donc on aura nécessairement une erreur si on essaye de l'effectuer sur un espace mémoire limité à un octet.

3. *Conversion entre bases de numération* (le détail des calculs est demandé – le résultat seul ne rapportera pas la totalité des points.)

(a) (1 point) Convertissez de base 10 en base 2: $26_{(10)}$

(b) (1 point) Convertissez de base 16 en base 2: $2B_{(16)}$

Solution: (a) On sait qu'il y a deux méthodes – je vous laisse revoir le cours pour celle dite des "soustractions successives" et je vous détaille ici celle des "divisions successives" que vous aviez préférée lors de nos exercices en cours:

$$\begin{array}{ll} 26/2 = 13 & \text{reste } 0 \\ 13/2 = 6 & \text{reste } 1 \\ 6/2 = 3 & \text{reste } 0 \\ 3/2 = 1 & \text{reste } 1 \\ 1/2 = 0 & \text{reste } 1 \end{array}$$

La conversion en base 2 de 26_{10} est la liste des restes de ces divisions prises de bas en haut: 11010_2.

(b) La conversion de base 16 en base 2 est assez simple – il suffit de convertir chacun des chiffres en base 2 et de mettre les résultats côte à côte (c'est un des intérêts de la base hexadécimale, dû au fait que 16 est une puissance de 2 – je vous renvoie au cours si cette notion n'est pas claire).

$2_{16} = 2_{10} = 10_2$ – ça, ça devrait être assez immédiat.

B_{16}, c'est 11 en base 10 – puisque lorsqu'on compte dans les deux bases cela donne:

$$\begin{array}{c} 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \underline{11}\ 12\ 13\ 14\ 15\ 16\ 17 \\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ \underline{B}\ C\ D\ E\ F\ 10\ 11 \end{array}$$

Reste donc à convertir 11 de base 10 en base 2 – pour cela on peut utiliser la méthode utilisée à l'exercice (a) précédent, ou bien tout simplement compter en base 2 jusqu'à 11:

$$\begin{array}{c} 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \underline{11} \\ 0\ 1\ 10\ 11\ 100\ 101\ 110\ 111\ 1000\ 1001\ 1010\ \underline{1011} \end{array}$$

Et donc si l'on met bout à bout ces deux conversions on obtient: $2B_{(16)} = 101011_{(2)}$.

4. *Entiers relatifs*

On considère le nombre binaire $10011100_{(2)}$.

(a) (1 point) Quelle est la valeur en base 10 de ce nombre, s'il représente un entier non signé sur un octet[1]?

(b) (1 point) Quel est le complément à un de 10011100?

(c) (1 point) Quel est le complément à deux de 10011100?

(d) (1 point) Quelle est la valeur en base 10 du nombre binaire 10011100 s'il représente un entier signé?

Solution: (a) On parle d'entier "non signé" – donc un entier naturel qui ne porte pas de bit de signe. On applique donc une stricte conversion de base 2 en base 10, en parcourant le nombre de droite à gauche et en lui appliquant les puissances de 2 successives:

$$0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 = 4 + 8 + 16 + 128 = 156$$

Donc: $10011100_{(2)} = 156_{(10)}$

[1] Au cas où ça pourrait vous être utile: $2^7 = 128$

(b) Le complément à un (voir le cours) est une simple inversion de tous les bits qui constituent le nombre – donc ici: 01100011

(c) Le complément à deux (voir, encore une fois, le cours) est un ajout de 1 au complément à un – donc ici:

$$\begin{array}{r} 0\,1\,1\,0\,0\,0\,1\,1 \\ + \qquad\qquad 1 \\ \hline 0\,1\,1\,0\,0\,1\,0\,0 \end{array}$$

(d) Si ce nombre 10011100 représente un entier signé, alors son complément à deux représente son opposé – c'est le principe-même de l'encodage des entiers relatifs qu'on a vu en cours. Donc puisqu'on sait que ce nombre est négatif (son bit de signe, bit de poids fort, vaut 1), il nous suffit de convertir de base 2 en base 10 son complément à deux que l'on vient de calculer (01100011) pour connaître sa valeur, en appliquant la même méthode qu'à la partie (a) de cette question:

$$0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 0 \times 2^7 = 4 + 32 + 64 = 100$$

Donc la valeur en base 10 du nombre binaire 10011100 s'il représente un entier signé est **-100**.

5. *Chiffrement*

On considère le code python ci-dessous[2].

```
1  message = ["M", "A", "X", "!"]
2  decal = 5
3  resultat = ""
4
5  # Processus de transformation
6  for i in range(len(message)):
7      caractere = message[i]
8      if ord("A") <= ord(caractere) <= ord("Z"):
9          code_ascii = ord(caractere)
10         print(code_ascii) # PRINT #1
11         nouveau_code = code_ascii + decal
12         nouveau_caractere = chr(nouveau_code)
13         print(nouveau_caractere) # PRINT #2
14         resultat = resultat + nouveau_caractere
15     else:
16         resultat = resultat + caractere
17
18 # Affichage final
19 print ("Resultat :", resultat ) # PRINT #3
```

(a) (2 points) Quel est l'affichage en console si l'on exécute ce code? (notez qu'il y a 3 print dans l'ensemble du code.) A quoi pourrait servir ce code?

On pourra s'aider de l'extrait de la table ascii ci-dessous :

65 : A	66 : B	67 : C	68 : D	69 : E	70 : F	71 : G	72 : H	73 : I	74 : J
75 : K	76 : L	77 : M	78 : N	79 : O	80 : P	81 : Q	82 : R	83 : S	84 : T
85 : U	86 : V	87 : W	88 : X	89 : Y	90 : Z	91 : [	92 : \	93 :]	94 : ^
95 : _	96 : ‘	97 : a	98 : b	99 : c	100 : d				

(b) (1 point) Quel est le problème avec la lettre "X"?

(c) (1 point) Comment pourriez-vous résoudre ce problème?

[2]Deux rappels: `ord("X")` renvoie le code ASCII du caractère "X"; `chr(88)` renvoie le caractère dont le code ASCII est 88.

Solution: (a) Je veux m'attarder un peu ici sur la ***méthode*** qu'il convient d'utiliser pour ce genre de question – et plus généralement quand vous êtes amenés à lire du code:

- Commencez par repérer les structures logiques majeures (conditions, boucles...) – en python c'est facile, puisqu'elles sont indentées. Regardez où elles commencent, où elles finissent – et isolez dans votre tête ce qu'elle font unitairement.
- Dans un second temps, exécutez pas à pas sur une feuille de papier (ou dans votre tête si c'est suffisamment simple) les premières étapes du code pour repérer sa logique. Dans le cas de ce code-ci, cela donne:
  - J'initialise mes variables puis je rentre dans la boucle qui parcourt la liste message.
    - i = 0, donc caractere = message[0] = "M"
    - ord(caractere) = 77 (d'après la table ASCII fournie) qui est bien entre ord("A") (65) et ord("Z") (90) – donc on rentre dans le if:
      - code_ascii = ord(caractere) = 77
      - print(code_ascii) → On a la première sortie dans la console: **77**
      - nouveau_code = code_ascii + decal = 77 + 5 = 82
      - nouveau_caractere = chr(nouveau_code) = chr(82) = "R" (toujours d'après la table ASCII fournie).
      - print(nouveau_caractere) → On a la deuxième sortie dans la console: **R**
      - resultat = resultat + nouveau_caractere → Comme on avait initialisé resultat à "", cela signifie que resultat vaut maintenant "R".
    - On est sorti du "if"
  - On a fini la première instance de la boucle "for", donc...
  - On recommence avec i = 1, donc caractere = message[1] = "A"

Arrêtons-nous là un instant et réfléchissons à ce que nous avons appris:

- Nous avons parcouru intégralement une instance de la boucle – donc tout ce qui va suivre ne sera que des répétitions de ce qu'on vient de faire.
- On a vu que si le caractère était inclus entre A et Z – ce qui est le cas des trois premiers de la liste message – on allait faire ce décalage de "decal" (soit 5) pour récupérer une nouvelle lettre dans la table ASCII.
- Donc on *sait* que pour chacun des deux caractères suivants "A" et "X" on va faire la même chose: ajouter 5 au code ASCII et produire le caractère correspondant.
- Donc à quoi sert ce code qui remplace systématiquement un caractère par un autre? Dans un exercice qui en plus s'appelle "chiffrement"? A coder / chiffrer un texte bien sûr!

Pour ce qui est de la suite en affichage console, on reproduit ce qu'on vient de faire avec les deux éléments suivants de la liste ("A" et "X"), puis avec le dernier ("!") on se rend compte qu'on n'est pas dans l'intervalle [ord("A"),ord("Z")], que donc on ne rentre pas dans le "if" – et qu'on ajoute donc directement le caractère "!" à resultat sans le coder avant de produire le print final. Au total cela donne:

```
77
R
65
F
```

```
88
]
Resultat : RF]!
```

Les questions suivantes sont beaucoup plus difficiles que ce que je prévoyais. Vous n'avez PAS besoin de les revoir pour le contrôle de remédiation.

(b) Le problème avec la lettre "X" est qu'en ajoutant 5 à son code ASCII on dépasse l'intervalle des lettres majuscules pour entrer dans celui des signes de ponctuation – dans ce cas, "]", ce qui si on est en train de crypter un texte risque de faire tâche et donner une indication à d'éventuels ennemis sur comment on a codé le texte.

(c) La solution la plus évidente consisterait à "reboucler" vers la valeur du code de la lettre A – quand on est à U (85), on passe à Z (90), puis quand on est à V (86) on passe à A (65) et ainsi de suite. Comment réaliser cela en pratique? En utilisant l'opérateur "%" (modulo):

```python
nouveau_code = code_ascii + decal
if nouveau_code > ord("Z"):
    nouveau_code = nouveau_code % ord("Z") + ord("A") - 1
```

6. *Parcours d'une liste*

On vous demande de coder une fonction CodeASCII(lst) qui prend en entrée une liste de caractères et renvoie en sortie une liste de même longueur contenant le code ASCII de chacun des caractères. Par exemple (en se référant à l'extrait de la table ASCII de l'exercice précédent) on aurait `CodeASCII(["M", "A", "X"]) = [77, 65, 88]`. *Note: il est évident que l'on se servira ici de la fonction ord() utilisée dans l'exercice précédent*[2].

(a) (2 points) Rédigez dans un premier temps l'algorithme qui sera mis en œuvre par une telle fonction – idéalement sous forme de pseudo-code.

(b) (2 points) Rédigez dans un second temps le code python de la fonction CodeASCII(lst).

Solution: Cet exercice est pratiquement identique à ceux que l'on a effectués en cours et dans la feuille d'exercices supplémentaires que je vous ai envoyée (recherche du maximum d'une liste, du minimum, calcul de la moyenne, etc.) – ce qui diffère, c'est ce que l'on fait *dans* la boucle, tandis que la réelle difficulté (le parcours de la liste) est strictement la même.

(a)

Entrée: ListeCar, liste de caractères individuels
Sortie: resultat, liste d'entiers correspondant à la valeur du code ASCII en entrée

```
fonction CodeASCII(chn)
    LongChn ← len(ListeCar)                    ▷ Nombre de caractères en entrée
    resultat ← [ ]                             ▷ Initialisation du résultat: liste vide
    pour tout caractere de ListeCar faire
        CodeCourant ← ord(caractere)           ▷ Récupération code ASCII
        resultat ← resultat + CodeCourant      ▷ Ajout en bout de liste
    fin pour
    retourner resultat
fin fonction
```

(b)

```python
def CodeASCII(lst):
    NbCar = len(lst)
    resultat = []
    for i in range(NbCar):
        CodeCourant = ord(lst[i])
        resultat.append(CodeCourant)
    return resultat
```

7. *Codage des flottants – norme IEEE 754 à simple précision*

L'objectif de cet exercice est de découvrir quel est le nombre réel codé par 11000000110100000000000000000000.

Ce nombre sera appelé N dans cet exercice.

On rappelle que le premier bit est `?????????????` (noté S), les 8 bits suivants correspondent à `?????????????` (noté E), et les 23 bits suivants à `?????????????` (noté M).

Ainsi, on a $S = 1$, $E = 10000001$ et $M = 10100000000000000000$.

On rappelle que $2^{-1} = 0.5$, $2^{-2} = 0.25$, $2^{-3} = 0.125$.

On rappelle la formule suivante : $N = (-1)^S \times (1 + M) \times 2^{E-127}$.

(a) (1 point) Écrivez sur votre feuille, dans le bon ordre, les mots qui doivent figurer à la place des trois "`?????????????`" dans l'énoncé de cet exercice.

(b) (½ point) N est-il positif ou négatif?

(c) (½ point) Quelle est la valeur de E?

(d) (½ point) Quelle est la valeur de M?

Solution:

(a)

- Bit de signe
- Exposant
- Mantisse

(b) Le bit de signe étant positionné à 1, on aura $(-1)^1 = -1$ et donc le nombre sera négatif.

(c) E, en binaire, vaut 10000001. Le calcul est donc simple puisqu'on n'a que la première et la dernière puissances de 2 qui sont positionnées à 1: $E = 2^0 + 2^7 = 1 + 128 = 129$

(d) M, en binaire, vaut 10100000000000000000. On n'a donc de nouveau que deux puissances de 2 qui sont positionnées à 1 – et dont l'énoncé nous donne la valeur, donc le calcul est immédiat: $M = 2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$

Et enfin la question (e) qui n'était pas posée dans le contrôle:

$$N = (-1)^1 \times (1 + 0.625) \times 2^{(129-127)} = -1 \times 1.625 \times 4 = -6.5$$

(Question bonus 1)

Code mystère: quel est l'affichage obtenu en console si on exécute ce code? Que signifie-t-il / que fait ce code? On ne demande pas de détailler les étapes du calcul, mais d'expliciter le lien entre les variables de départ et ce qui est affiché à la fin du programme. (conseil: commencez par exécuter ce programme "à la main" pour voir ce qui se passe à chaque étape)

```
1  x = 97
2  puissance_2 = 2**7 #(ce qui vaut 128)
3  res = ""
4  while puissance_2 >= 1:
5      if puissance_2 <= x:
6          res += "1"
7          x = x - puissance_2
8      else:
9          res += "0"
10     puissance_2 = puissance_2 / 2
11 print (res)
```

Solution: Cet exercice est plus compliqué que le reste du contrôle mais il reste très abordable et la plupart d'entre vous devrait être capable de le faire – je vous invite donc à bien étudier sa correction.

Commençons par exécuter pas à pas les premières étapes de ce code pour essayer de comprendre ce qu'il se passe:

Etape	Ligne	Action
1	1	x = 97
2	2	puissance_2 = 128
3	3	res = ""
4	4	puissance_2 = 128 $\geq$ 1, donc on rentre dans la boucle
5	5	puissance_2 = 128 > 97 = x, donc on ne rentre pas dans le if
6	8	else
7	9	res += "0" donc res = "0"
8	10	puissance_2 = puissance_2 / 2 = 128 / 2 = 64 ($= 2^6$)
9	4	puissance_2 = 64 $\geq$ 1, donc on va effectuer une nouvelle itération de la boucle
10	5	puissance_2 = 64 $\leq$ 97 = x, donc on rentre dans le if
11	6	res += "1" donc res = "01"
12	7	x = x – puissance_2 = 97 – 64 = 33
13	10	puissance_2 = puissance_2 / 2 = 64 / 2 = 32 ($= 2^5$)
14	4	puissance_2 = 32 $\geq$ 1, donc on va effectuer une nouvelle itération de la boucle
15	5	puissance_2 = 32 $\leq$ 33 = x, donc on rentre dans le if
16	6	res += "1" donc res = "011"
17	7	x = x – puissance_2 = 33 – 32 = 1
18	10	puissance_2 = puissance_2 / 2 = 32 / 2 = 16 ($= 2^4$)

Quelques constats:

- On voit que "res", qui est la valeur qui va être affichée à l'utilisateur en fin de programme (et qui donc est a priori le "but" de ce programme), est une chaîne de caractères qui ne peut contenir *que* des "0" (ligne 9) et des "1" (ligne 6).
- On voit que l'on fait diminuer x progressivement en en soustrayant toujours une puissance de 2 (ligne 7): ça, on le sait parce que la variable s'appelle "puissance_2" mais aussi parce qu'elle est initialisée à une puissance de 2 (ligne 2) et n'est par la suite modifiée *que* par division par 2 (ligne 10).

Prenez ces constats, ajoutez-y le fait que le thème principal du contrôle est la représentation de données et notamment le passage d'une base à une autre et il est évident que ce programme convertit un entier décimal x (de valeur 97 en l'occurrence) en binaire sous forme d'une chaine de caractères

(des "1" et des "0" successifs). On peut noter que l'on commence par un "0" puisque la première puissance de 2 que "regarde" le programme est 2^7 qui est supérieure à la valeur de x.

L'affichage en console sera donc la conversion en base 2 de 97:

```
01100001
```

(Question bonus 2)

En cours on a codé une fonction CompUn(lst) qui renvoie le complément à un d'une liste de bits représentant un entier codé en binaire. On vous demande de coder une deuxième fonction, qu'on appellera Ajouter1(lst), qui prendra le résultat de la précédente, ajoutera un, et renverra donc le complément à deux de l'entier initial.

Quelques remarques:

- La syntaxe pour une boucle bornée dont l'indice va descendre de N à 0 est `for i in range(N,-1,-1):`
- On ignorera le cas d'une liste uniquement composée de 1 (et pour laquelle un ajout de un ajouterait un chiffre).
- Conseil: commencez par faire à la main `100111 + 1` et réfléchissez aux étapes que vous accomplissez, à comment vous gérez les retenues, à ce qui se passe quand il n'y a plus de retenue...

Solution: Cet exercice est *nettement* plus compliqué que le reste du contrôle: je vous invite bien évidemment à étudier sa correction, mais ne vous inquiétez pas s'il vous semble trop difficile – il l'est.

On commence par appliquer le conseil et on exécute à la main `100111 + 1`:

$$\begin{array}{r} 1\,0\,0\,1\,1\,1 \\ +\qquad\quad 1 \\ \hline 1\,0\,1\,0\,0\,0 \end{array}$$

Dans le détail, ce qu'on effectue comme opérations est:

1. On affecte à "somme" l'ajout de 1 au chiffre le plus à droite: le résultat est nécessairement inférieur à 2;
2. Si somme est ≤ 1 (ce qui n'est ***pas*** le cas ici):
   - On place "somme" à la droite du résultat et on passe à la suite: en pratique, on a terminé – aucun des autres chiffres ne sera modifié puisqu'on n'ajoute que 1 en tout.
3. Sinon (si somme = 2 – c'est notre cas):
   - On convertit le résultat en binaire – $2_{10} = 10_2$
   - On écrit donc 0 à droite du résultat, et on fait une retenue de 1.

Ces étapes nous ont permis d'écrire le chiffre le plus à droite du résultat; pour écrire les suivants, on reprend exactement le même raisonnement, en modifiant simplement la toute première étape – "Ajout de 1 au chiffre le plus à droite" devient "Ajout de l'éventuelle retenue au chiffre suivant".

Dès lors, si on a une retenue on réitère le même raisonnement, et dès qu'on n'en a plus (à partir du 5ème chiffre dans notre exemple) on n'a plus qu'à "faire descendre" les chiffres suivants dans le résultat, jusqu'au dernier.

Attention: si on a "ajouté" les chiffres au résultat en utilisant la méthode "append", on les a placés de gauche à droite. Il faudra donc inverser l'ordre de la liste avant de la renvoyer à la fin de la fonction.

En pseudo-code, ceci se traduit par:

Entrée: NbBin, nombre binaire sous forme de liste de 0 et de 1
Sortie: resultat, nombre binaire sous forme de liste de 0 et de 1

```
1:  fonction AJOUTER1(NbBin)
2:      ProchainAjout ← 1        ▷ Ajout à effectuer à la prochaine étape: 1, puis retenue éventuelle
3:      resultat ← [ ]                          ▷ Initialisation du résultat: liste vide
4:      pour tout chiffre de NbBin de droite à gauche faire
5:          Somme ← chiffre + ProchainAjout
6:          si Somme ≤ 1 alors                  ▷ Cas où il n'y aura pas de retenue
7:              resultat ← resultat + Somme     ▷ Ajout du chiffre calculé à la liste résultat
8:              ProchainAjout ← 0               ▷ Pas de retenue
9:          sinon
10:             resultat ← resultat + 0         ▷ Le résultat était 2, donc 10 en binaire
11:             ProchainAjout ← 1               ▷ Retenue
12:         fin si
13:     fin pour
14:     Inverser l'ordre des éléments de resultat (ce qui en python peut se faire en une commande)
15:     retourner resultat
16: fin fonction
```

Et enfin cette fonction, traduite en Python, donne:

```python
def Ajouter1(lst):
    ProchainAjout = 1
    resultat = []
    for i in range(len(lst) - 1, -1, -1):
        Somme = ProchainAjout + lst[i]
        if Somme <= 1:
            resultat.append(Somme)
            ProchainAjout = 0
        else:
            resultat.append(0)
            ProchainAjout = 1

    # Méthode qui inverse l'ordre d'une liste
    # (note: on aurait évidemment pu faire ça à la main par le biais d'une boucle for)
    resultat.reverse()

    return resultat
```