1ère Spé. NSI Gr. 2 **Contrôle #3, 08/01/2024**
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. *Questions de cours:*
   (a) (1 point) Expliquez (en une ou deux phrases) en quoi l'encodage UTF-8 est plus performant que son ancêtre, le codage ASCII, et est donc aujourd'hui presqu'universellement utilisé par exemple pour les sites web.

   Solution:
   Le code ASCII, le premier système d'encodage de caractères alphanumériques en binaire, utilisait 7 bits et permettait donc l'encodage de 128 caractères (2^7). Il était donc extrêmement limité et n'autorisait ni les accents, ni encore moins les alphabets ou syllabaires autres que l'alphabet latin – l'arabe, le cyrillique, l'hébreu, l'amharique... L'UTF-8, créé dans les années 1990, permet lui d'utiliser un nombre variable de bits pour encoder un caractère pouvant aller jusqu'à 24 bits. Il peut représenter jusqu'à 1 million de caractères différents ce qui le rend idéal pour les applications internationales, y compris les pages Web.

   (b) (1 point) Quel est l'entier naturel (donc appartenant à l'ensemble $\mathbb{N}$) maximal que l'on peut coder sur 4 bits? (vous donnerez le détail du calcul)

   Solution:
   Sur 4 bits, le nombre maximal que l'on peut coder est 1111_2 soit, en décimal: $2^0 + 2^1 + 2^2 + 2^3 = 15$. Mais on peut effectuer un calcul plus simple: en effet $(1111 + 1)_2 = 10000_2$; donc $1111_2 = 10000_2 - 1 = 2^4 - 1 = 16 - 1 = 15$.

   (c) (1 point) Quel est l'entier relatif positif (donc appartenant à l'ensemble $\mathbb{Z}$) maximal que l'on peut coder sur 4 bits? (vous donnerez le détail de votre raisonnement)

   Solution:
   Si on veut stocker des entiers relatifs sur 4 bits, il faut stocker les nombres positifs *et* les nombres négatifs, donc on peut en stocker moitié moins que d'entiers naturels. On peut donc stocker 8 entiers négatifs, et 7 positifs, ce qui fait bien un total de 15 (le 16ème nombre étant le 0). La réponse à la question est donc 7 – mais 8 était également accepté (puisque -8 est le plus petit nombre négatif que l'on peut coder sur 4 bits).

2. *Addition de nombres binaires* (posez bien votre addition – un simple résultat ne sera pas accepté.)
   (a) (1 point) Effectuez l'addition suivante: 11101011 + 01001111
   (b) (1 point) Si ces deux nombres binaires représentaient des entiers signés (ou entiers relatifs appartenant à l'ensemble $\mathbb{Z}$), quel serait le signe du résultat? Justifiez votre réponse.

Solution:

		$^{1}1$	1	1	$^{1}0$	$^{1}1$	$^{1}0$	$^{1}1$	1
+		0	1	0	0	1	1	1	1
	1	0	0	1	1	1	0	1	0

La dernière retenue fait que l'on a un resultat comportant 9 bits; or on sait que lorsqu'on additionne deux entiers relatifs codés sur n octets, s'il y a une retenue finale on l'ignore (comme on l'a vu en cours, c'est grâce à cette "astuce" que la méthode du complement à 2 fonctionne) et on ne retient donc que les n bits de droite – ici, puisque l'on est sur un octet, les 8 bits de droite: `00111010`. On regarde alors le bit de poids fort (le plus à gauche) de ce resultat pour déterminer le signe, et ici, puisque ce bit est à 0, on conclut que le résultat de cette addition est un entier positif.

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 hexadécimale (16) en base décimale (10): $2AF_{(16)}$ (on rappelle que $16^2 = 256$)
   (b) (1 point) Convertissez de base 10 en base 2: $37_{(10)}$

Solution:

(a) On sait que chacun des "chiffres" de ce nombre représente le multiple d'une puissance de 16 puisque nous sommes en base hexadécimale.

On sait par ailleurs compter en hexadécimal de 0 à 15:

0 1 **2** 3 4 5 6 7 8 9 **A** *B C D E* **F**
0 1 **2** 3 4 5 6 7 8 9 **10** 11 12 13 14 **15**

Il ne reste plus qu'à effectuer la somme des puissances de 16 correspondantes:

$2AF_{16} = (2\times16^2+10\times16^1+15\times16^0)_{10} = (2\times256+10\times16+15\times1)_{10} = (512+160+15)_{10} = 687_{10}$

(b) 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:

$37/2 = 18$ reste 1
$18/2 = 9$ reste 0
$9/2 = 4$ reste 1
$4/2 = 2$ reste 0
$2/2 = 1$ reste 0
$1/2 = 0$ reste 1

La conversion en base 2 de 37_{10} est la liste des restes de ces divisions prises de bas en haut: 100101_2.

4. *Entiers relatifs*

   On considère le nombre binaire $10011010_{(2)}$.
   (a) (1 point) Quelle est la valeur en base 10 de ce nombre, s'il représente un entier non signé (ou entier naturel appartenant à l'ensemble $\mathbb{N}$) sur un octet[1]?
   (b) (1 point) Quel est le complément à un de 10011010?
   (c) (1 point) Quel est le complément à deux de 10011010?
   (d) (1 point) Quelle est la valeur en base 10 du nombre binaire 10011010 s'il représente un entier signé (ou entier relatif appartenant à l'ensemble $\mathbb{Z}$)?

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

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:

$$0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 = 2 + 8 + 16 + 128 = 154$$

Donc: $10011010_{(2)} = 154_{(10)}$

(b) Le complément à un (voir le cours) est une simple inversion de tous les bits qui constituent le nombre – donc ici: 01100101

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

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

(d) Si ce nombre 10011010 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 (01100110) pour connaître sa valeur, en appliquant la même méthode qu'à la partie (a) de cette question:

$$0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 0 \times 2^7 = 2 + 4 + 32 + 64 = 102$$

Donc la valeur en base 10 du nombre binaire 10011010 s'il représente un entier signé est **-102**.

5. (2 points) *Chiffrement*

   On considère le code python suivant[2].

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

```python
message = ["N", "S", "I", "!"]
decal = 3
resultat = ""

# Processus de transformation
for i in range(len(message)):
    caractere = message[i]
    if ord("A") <= ord(caractere) <= ord("Z"):
        code_ascii = ord(caractere)
        print(code_ascii) # PRINT #1
        nouveau_code = code_ascii + decal
        nouveau_caractere = chr(nouveau_code)
        print(nouveau_caractere) # PRINT #2
        resultat = resultat + nouveau_caractere
    else:
        resultat = resultat + caractere

# Affichage final
print ("Resultat :", resultat ) # PRINT #3
```

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				

Solution:

C'est évidemment exactement le même exercice qu'au contrôle précédent, au détail près que le texte en entrée diffère. Je ne reviens donc pas en détails ici sur les raisonnements – je vous renvoie pour celà à la correction du contrôle précédent.

Ce que va afficher ce code est:

```
78
Q
83
V
73
L
Resultat : QVL!
```

6. *Parcours d'une liste*

On vous demande de coder une fonction `TempSeuil(lst, seuil)`:

- Elle prend en entrée deux arguments:
  1. Une liste d'entiers représentant des températures (`lst`)
  2. Un entier représentant une température seuil (`seuil`)
- Elle renvoie en sortie une liste de toutes les températures présentes dans la liste fournie en entrée qui sont strictement supérieures au seuil.

Par exemple on aurait `TempSeuil([-2, 10, 15, 8, 17], 10) = [15, 17]`. *Note: la valeur 10 n'est pas présente en sortie puisqu'on considère les valeurs strictement supérieures au seuil uniquement.*

(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 `TempSeuil(lst, seuil)`.

Solution:

Cet exercice est pratiquement identique à ceux que l'on a effectués en cours, dans la feuille d'exercices supplémentaires que je vous ai envoyée (recherche du maximum d'une liste, du minimum, calcul de la moyenne, etc.), et dans le contrôle précédent. Il s'agit de parcourir une liste et d'effectuer un traitement simple (ici un test pour voir si la valeur considérée est strictement supérieure à un seuil) pour chacune des valeurs qui la composent.

(a)

Entrée: ListeTemp, liste de températures sous forme d'entiers; Seuil, un entier
Sortie: Resultat, liste d'entiers correspondant aux températures supérieures au seuil

```
1:  fonction TempSeuil(ListeTemp, Seuil)
2:      LongChn ← len(ListeTemp)          ▷ Nombre de températures en entrée
3:      Resultat ← [ ]                    ▷ Initialisation du résultat: liste vide
4:      pour tout temp de ListeTemp faire
5:          si temp > Seuil alors
6:              Resultat ← Resultat + temp          ▷ Ajout en bout de liste
7:          fin si
8:      fin pour
9:      retourner Resultat
10: fin fonction
```

(b)

```python
def TempSeuil(lst, seuil):
    NbCar = len(lst)
    resultat = []
    for i in range(NbCar):
        if lst[i] > seuil:
            resultat.append(lst[i])
    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:

11000001010110000000000000000000

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 $S = 1$, $E = 10000010$ et $M = 01100000000000000000$.

On rappelle que $2^{-1} = 0.5$, $2^{-2} = 0.25$, $2^{-3} = 0.125$, $2^{-4} = 0.0625$ (*mais vous pouvez vous passer de cela pour répondre aux questions – sauf si vous préférez les calculs compliqués)...*).

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) (1 point) Quelle est la valeur de N (il n'est pas obligatoire de calculer la valeur exacte de M, juste celle de N; on rappelle que $x^a \times x^{-b} = x^{a-b}$ quels que soient les entiers a et b)?

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 10000010. Le calcul est donc simple puisqu'on n'a que la première et l'avant-dernière puissances de 2 qui sont positionnées à 1: $E = 2^1 + 2^7 = 2 + 128 = 130$

(d) Vous pouvez vous amuser à calculer la valeur exacte de M si vous voulez (elle est de $2^{-1} + 2^{-3} + 2^{-4} = 0,6875$) mais en utilisant son developpement en puissances de 2 le calcul devient extrêmement simple:

$$\begin{aligned} N &= (-1)^1 \times (1 + 2^{-1} + 2^{-3} + 2^{-4}) \times 2^{130-127} \\ &= (-1) \times (1 \times 2^3 + 2^{-1+3} + 2^{-3+3} + 2^{-4+3}) \\ &= (-1) \times (8 + 4 + 1 + \frac{1}{2}) \\ &= -13,5 \end{aligned}$$

(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)

```
x = 97
puissance_2 = 2**7 #(ce qui vaut 128)
res = ""
while puissance_2 >= 1:
    if puissance_2 <= x:
        res += "1"
        x = x - puissance_2
    else:
        res += "0"
    puissance_2 = puissance_2 / 2
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
```