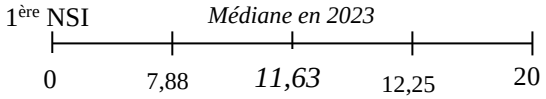
CORRIGE CONTROLE 5 (55')
ECRITURE BINAIRE ; NUMERISATION

Compte rendu : Il s'agissait de la copie quasi-conforme du sujet de l'année dernière sauf l'exo 2 !

- *Exo 1 : Beaucoup de questions sont issues des sujets des années précédentes.*
  ***Le cours n'est pas su (2022)**. L'inversé bit à bit + 1 n'est pas connu de certains !*
- *Exo 2 : Très peu réussi en 2023.*
- *Exo 3 : Similaire à l'additionneur fait en TP et contrôle 2022.*
  Question 1-2-3 : Questions moyennement réussies en 2023.
  Question 4 : Catastrophique pour certains ou tout simplement non fait. Pourtant cet exo a été travaillé pour le TP et est dans le contrôle 2022 ⇒ cela pose la question de comment l'évaluation est préparée.



Traitée seulement 1 fois sur 11 correctement en 2023.

- *Plus généralement : Bien lire les détails de ce corrigé. Apprendre son cours, faire et refaire les quizz de préparation jusqu'à ce qu'ils soient parfaitement réussis, et les évaluations des années précédentes, refaire les exos de programmation du cours !*
- *Programmer de choses simples est hors de portée de la majorité.*

Médianes = 12,25 en 2022.

➢ Exercice n° 1 (……..…… / 12 pts) : Ecriture binaire, Numérisation des nombres et des textes.

① Une image est constituée de 4 000 points. Chaque point a sa couleur codée sur 32 bits.

Quel est le poids de cette image en ko (kilo-octets) ? Solution correctement rédigée. (…… / 1,5 pts)

$$\textit{Poids de l'image(en ko)} = \frac{\textit{Nb de points} \times \textit{Poids d'un point en bits}}{\textit{Nb de bits dans 1 ko}}$$

$$= \frac{4\,\cancel{000} \times 32}{8 \times 1\,\cancel{000}}$$

$$= 16 \text{ ko}$$

L'image pèse 16 ko.

Solution très souvent mal rédigée : Analyse-Synthèse ! Une rédaction n'est pas un brouillon !

② Pour convertir en base 10 un nombre écrit en base 2, on peut :

1. effectuer une suite de divisions euclidiennes par 2.
2. effectuer une suite de divisions euclidiennes par 10.
3. additionner des puissances de 2.
4. additionner des puissances de 10.

*Que d'erreurs à cette question ! **Vérifier en prenant un exemple tout bêtement !***

A retenir : (base p → base 10) : additionner des puissances de p.

(base 10 → base p) : suite de divisions euclidiennes par p.

③ Toutes les réponses sont sur 0,5 points.

- Ecrire l'intervalle d'entiers (*non signés*) codables sur N bits : $[\![0 ; 2^N - 1]\!]$
- Ecrire le nombre d'entiers (*non signés*) codables sur 5 bits : $2^5 = 32$
- Calculer la valeur de l'entier non signé dont l'écriture binaire est 10110 :

$(10110)_2 = 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 22.$

④ Le plus petit entier **signé négatif** représentable sur N bits est :

Cours Livret Numérisation des nombres p.15 !

1. -2^N
2. $-2^N - 1$
3. -2^{N-1}
4. $-2^{N-1} - 1$

⑤ L'écriture binaire 10110 est celle de l'entier ***signé*** :

Ecrire l'inversé bit à bit de 10110 →

Rajouter 1 à cet inversé bit à bit →

Donner la valeur de cet inversé bit à bit + 1 →

Prendre l'opposé →

1. -9 *Oubli du rajouter 1.*
2. -10
3. 22 *C'est l'entier non signé !*
4. -6 *Il ne suffit pas de transformer le 1er 1 en signe – et compter le reste !*

⑥ Donner l'écriture binaire sur 4 bits de l'entier **signé** -7. (..... / 1,5 pts)

Ecriture binaire de 7 sur 4 bits.

Prendre l'inversé bit à bit.

Rajouter 1 à cet inversé bit à bit. 1001

Méthode par complément à 2^n

Rajouter 2^n au nombre → $-7 + 2^4 = -7 + 16 = 9$

Donner l'écriture binaire de 9 → 1001

Solution très souvent mal rédigée ! Une rédaction n'est pas un brouillon !

⑦ Donner l'écriture hexadécimale de 11010101 : ……

Chaque groupe de 4 bits est remplacé par un chiffre hexadécimal. D5.

Valeur de $(2A)_{16}$ *= $2 \times 16^1 + 10 \times 16^0$*

= 32 + 10 = 42

⑧ A l'aide de la table ASCII fournie avec le manuel d'une imprimante de 1972, donner :

(…….. / 0,5 + 0,5 pts)

- l'écriture binaire du caractère « G » :

Il suffit de lire les bits b7 à b1 à l'intersection de la colonne 4 et de la ligne 7 : 100 0111.

- le point de code du caractère « + » :

Ecriture binaire par l'intersection de la colonne 2 et la ligne 11 : 010 1011.

Puis calculer : $(010\ 1011)_2 = (43)_{10}$

Autre façon : colonne 2 et ligne 11 → $(2C)_{16} = 2 \times 16 + 11 = 43$

Autre façon : compter à partir du caractère NUL qui a pour point de code 0 !

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	¦
1 1 0 1	13	CR	GS	—	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

⑨ Soit le contenu suivant d'un mail reçu : « BÃ©bÃ© fait du sale, allÃ´ allÃ´ allÃ´

Million d'dollars, bÃ©bÃ© tu vaux Ã§a »

- Quel(s) caractères posent des problèmes ? (…….. / 0,25 pts)

Les caractères qui posent problème sont « é », « ô » et « ç ».

On a reconnu Pookie la chanson d'Aya Nakamura sortie le 10/4/2019.

- Comment expliquez-vous ce problème ? (……. / 0,5 pts)

Le charset utilisé à l'écriture du message n'est pas le même que celui utilisé à la lecture.

- Expliquer pourquoi le charset ASCII (non étendu) n'a logiquement pas été utilisé lors de l'écriture ou de la lecture de ce message. (…… / 0,25 pts)

Le charset ASCII ne permet ni de lire ni d'écrire des caractères accentués ou avec signe diacritique.

⑩ On sait que le point de code de la lettre 'a' est 97. Compléter le script suivant permettant d'afficher à l'écran les lettres minuscules de 'a' à 'h'. (..... / 1,5 pts)

Il y a 8 lettres de 'a' inclus à 'h' inclus.

```
for k in range(8) :
    print( chr(97 + k) )
```

Beaucoup ne connaissent pas la fonction chr( ). Beaucoup de fautes dans le range !

On pouvait aussi utiliser un range(97 + 0 , 97 + 7 + 1) avec un chr(k) mais alors beaucoup de fautes dans la borne sup du range( ) !

➢ Exercice n° 2 (……..…… / 3 points) : Ordre lexicographique.

• Rappel : L'ordre lexicographique est utilisé pour comparer 2 chaînes composées de n'importe quels caractères (lettres, chiffres, ponctuation, emojis etc.) et s'appuie sur la table Unicode

L'algorithme est le suivant : les 2 chaînes de caractères sont comparées caractère par caractère (ordre unicode) : le 1er avec le 1er, puis suivant à suivant tant qu'il y a égalité, en n'oubliant pas que tout caractère est supérieur au caractère vide (utile si une chaîne est plus grande que l'autre).

Ainsi par exemple on aura selon cet ordre lexicographique : 'f' > 'F' 'ab' > 'aa' 'abcd' > 'abc'.

• Ecrire en Python une fonction comparaison_lexicographique( ) avec :

- Entrées : chaine1 et chaine2 deux chaînes de caractères quelconques.
- Corps : algorithme lexicographique. Utilisation obligatoire de la fonction ord( ) pour comparer les caractères entre eux.
- Sortie : 1 si 'chaine1' > 'chaine2' , 2 si 'chaine2' > 'chaine1' , 0 si 'chaine1' == 'chaine2'.

Analyse algorithmique :

• On va comparer caractère par caractère et tant qu'il y a égalité, on passera aux suivants. On peut faire ces comparaisons autant de fois qu'il y a de caractères dans la plus petite des deux chaînes.

• S'il n'y a pas assez de caractères pour comparer, la chaîne la plus longue est la supérieure.

Synthèse Programmation :

```
1. def comparaison_lexicographique(chaine1 : str, chaine2 : str) -> int :
2.     longueur_minimum = min(len(chaine1),len(chaine2))
3.     for k in range(longueur_minimum) :
4.         if ord(chaine1[k]) > ord(chaine2[k]) :
5.             return 1
6.         if ord(chaine1[k]) < ord(chaine2[k]) :
7.             return 2
8.     else : # ce qui suit n'est exécuté que si la boucle for a été menée jusqu'à son terme
9.            # c-à-d quand les 2 chaînes sont parfaitement identiques jusqu'à la longueur min.
10.        if len(chaine1) > len(chaine2) :
11.            return 1
12.        elif len(chaine1) < len(chaine2) :
13.            return 2
14.        else : # len(chaine1) == len(chaine2) :
15.            return 0
```

Remarques :

- *Dans l'instruction composée For .. else, ce qui est dans le else n'est exécuté que si la boucle for a bien été mené jusqu'à son terme.*
- *On aurait pu utiliser une boucle Tant que à la place de la boucle For-If-Break, avec comme condition de poursuite : tant que caractère1[k] == caractère2[k] et k <= longueur_minimum.*

➢ Exercice n° 3 (……..…… / 5 points) : Soustraction binaire.

Les soustractions binaires se calculent chiffre à chiffre de la même façon qu'en décimale. Seule différence :

- lorsqu'on trouve -1, on pose 1 et on écrit dans la colonne à gauche une retenue *soustractive* de 1.
- lorsqu'on trouve -2, on pose 0 et on écrit dans la colonne à gauche une retenue *soustractive* de 1.

1. A droite, poser puis calculer la différence binaire suivante (mettre la ou les retenues en rouge) : 1011 – 0101

   Vérifiez en réadditionnant 0101 et votre différence trouvée !

   (……. / 0,5 pts)

```
 -1
  1 0 1 1
– 0 1 0 1
---------
  0 1 1 0
```

2. Soustraire 2 nombres binaires revient à enchaîner plusieurs soustractions élémentaires bit à bit (chiffre à chiffre) en tenant compte d'une éventuelle retenue à soustraire.

   Le tableau ci-dessous indique tous les résultats possibles pour la soustraction bit à bit en fonction des valeurs possibles pour bit1 le bit du premier nombre, bit2 le bit du second nombre à soustraire et l'éventuelle retenue à enlever.

   - Expliquer pourquoi ce tableau contient 8 colonnes à remplir : (…….. / 0,5 pts)

   bit1 : 2 choix (0 ou 1)
   bit2 : 2 choix (0 ou 1)
   retenue : 2 choix (0 ou 1)

   } *Les choix étant indépendants, ils se multiplient entre eux.*
   Donc 8 combinaisons possibles. Donc 8 colonnes.

   - Terminer de compléter les colonnes du tableau. (…….. / 1 pt)

Il faut faire le calcul bit1 – bit2 – retenue soustractive. Si on trouve -1, on pose 1 on retient 1. Si on trouve -2, on pose 0 et on retient 1.

bit1	0	1	0	1	*0*	*1*	*0*	*1*
bit2	0	0	1	1	*0*	*0*	*1*	*1*
retenue ***soustractive***	0	0	0	0	*1*	*1*	*1*	*1*
bit différence	0	1	1	0	*1*	*0*	*0*	*1*
retenue ***soustractive*** *à reporter*	0	0	1	0	*1*	*0*	*1*	*1*

3. Ecrire le dictionnaire Soustracteur_bit_à_bit correspondant à ce tableau. Ce dictionnaire ne doit contenir que des tuples de 'nombres' de type str. (…….. / 0,5 pt)

Ce dico servira à programmer la soustraction de 2 nombres binaires. Ces 2 nombres binaires en entrée seront de type str, ainsi que la différence binaire en sortie. ⇒ Que des str dans le dico !

Dico sous la forme : soustracteur_bit_à_bit = {('bit1' , 'bit2' , 'retenue soustractive') : ('bit_différence' , 'retenue_à_reporter')}

Soustracteur_bit_à_bit = { ('0','0','0') : ('0','0') ,
('1','0','0') : ('1','0') ,
('0','1','0') : ('1','1') ,
('1','1','0') : ('0','0') ,
('0','0','1') : ('1','1') ,
('1','0','1') : ('0','0') ,
('0','1','1') : ('0','1') ,
('1','1','1') : ('1','1') }

4. Ecrire une fonction soustraction_binaire( ) avec comme spécifications : (…… / 2,5 pts)

- Paramètres d'entrée : nbbinaire1 et nbbinaire2, deux nombres binaires de type str et de même longueur.
- Corps de la fonction : utiliser obligatoirement le dictionnaire précédent.
- Sortie : différence_binaire, la chaîne de caractères correspondant à la soustraction de nbbinaire1 par nbbinaire2.

```
1.  # coding : utf-8
2.  def soustraction_binaire(nb_binaire1 : str , nb_binaire2 : str)->str :
3.      # table de vérité de la soustraction bit à bit sous la forme d'un dico {('bit1','bit2', 'retenue
    soustractive') : ('bit_différence','retenue_à_reporter')}
4.      soustracteur_bit_à_bit={('0','0','0'):('0','0'),
5.                              ('1','0','0'):('1','0'),
6.                              ('0','1','0'):('1','0'),
7.                              ('1','1','0'):('0','1'),
8.                              ('0','0','1'):('1','0'),
9.                              ('1','0','1'):('0','1'),
10.                             ('0','1','1'):('0','1'),
11.                             ('1','1','1'):('1','1') }
12.     retenue_soustractive = '0'
13.     différence_binaire =''
14.     for k in range(len(nb_binaire1)):
15.         (bit_différence,retenue_à_reporter)=soustracteur_bit_à_bit[nb_binaire1[-1-k],nb_binaire2[-1-
    k],retenue_soustractive] # indice -1-k car on commence par le bit à droite de poids le plus faible.
16.         différence_binaire = bit_différence + différence_binaire
17.         retenue_soustractive = retenue_à_reporter
18.     if retenue_soustractive=='1' :   # gestion du débordement
19.         print('Il y a eu débordement')
20.     return(différence_binaire)
```