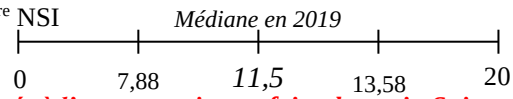
CORRIGE CONTROLE 1 (55') PROGRAMMATION DE BASE EN PYTHON

Compte rendu : Résultats très hétérogènes en 2019.



- *Les exos sur France IOI doivent être faits sinon à faire pendant l'évaluation !*
- ***Contrôle souvent mal préparé : cours non su (exo 1) ; exercices n°3-4 annoncés à l'avance mais non faits chez soi ; Sujets 0 et 0bis même pas lus. Beaucoup apparemment se contentent de lire le cours !***
- *Représentation des entiers (exo 1) : Raté dans l'ensemble. Complémentaire à 2^n inconnue pour les entiers signés.*
- *Structures de base de contrôle du flux (boucles, tests..) (exo 2) : Difficultés pour les boucles imbriquées → utilisez un tableau d'état des variables.*
- *Programmation (exos 3-4) : Catastrophique pour certains. Même si ces exos n'ont pas été préparés (anormal !), ce n'est quand même pas la première fois que vous écrivez un programme (cf France IOI où beaucoup d'exos sont bien plus durs !) !*

  Attention à la syntaxe de base : indentation, chaîne de caractères entre ' ', = = , etc.

  Test d'un programme : autant de valeurs que de parties issues du conditionnement.

➢ Exercice n° 1 (……..…… / 7 points) : QCM types entiers ; syntaxe de base.

- Compléter (aucun détail de calcul demandé) : (……. / 2 pts)

0010 1101 = *1 + 4 + 8 + 32 = 45* *Une des rares choses bien réussies.*

-7 par complémentaire à 2^n sur 4 bits = ? *Beaucoup d'élèves mettent la plage d'entiers sur n bits !*

+7 = 0111 → inversé bit à bit = 1000 → donc -7 = 1000 + 0001 = 1001

- Pour chaque affirmation, 4 choix vous sont proposés dont un seul est vrai. Lequel ? **L'entourer.**

Barème : réponse juste =1 pt sans réponse ou réponse fausse = 0 pt

Affirmations	*Choix 1*	*Choix 2*	*Choix 3*	*Choix 4*
*① Combien d'entiers **non signés** peut-on représenter en machine sur 32 bits ?*	2^{31}	2^{32}	32^2	$2^{32} - 1$
*② Sur 3 bits, le plus grand entier **signé positif** représentable en machine est*	$+2^{3-1} - 1$	$+2^{3-1}$	$+2^3 - 1$	$+2^3$
*③ L'entier **signé** 1111 vaut*	+15	-7	-1	+1
④ L'instruction qui n'est pas une auto-affectation est	k = 3 * k	k = k – 3	k = k + 3	k + 3 = k
⑤ L'instruction qui illustre une incrémentation de 2 est	x = y + 2	x + 2 = x	x = x + 2	x = 2 * x

① Sur n bits, l'intervalle d'entiers non signés représentables est $[\![0; +2^n - 1]\!]$ ce qui fait 2^n entiers, en comptant le 0 !

Beaucoup de confusions entre le nombre d'entiers non signés possibles (2^n) et le plus grand entier parmi eux ($2^n - 1$) !

② Sur n bits, l'intervalle d'entiers non signés représentables est $[\![-2^{n-1}; +2^{n-1} - 1]\!]$.

③ Très peu réussi en 2019.

Il faut chercher d'abord l'opposé par complémentaire à 2^n de 1111. On trouve ……. qui vaut …. Donc 1111 représente ….

+15 c'est quand on considère 1111 bêtement comme un entier non signé !

-7 c'est quand on change juste le signe et que l'on considère la distance à 0 comme la valeur des autres chiffres 111. Ce n'est pas le complémentaire à 2^n ! D'ailleurs le codage binaire de -7 par complémentation à 2^n a été trouvé normalement à la question 2 avant le tableau.

④ Une auto-affectation est une l'assignation d'une variable à elle-même. L'une des instructions affichera SyntaxError, laquelle ?

⑤ Une incrémentation est une auto-affectation additive (soustractive pour une décrémentation). (L'une des instructions affichera SyntaxError, laquelle ? Le choix …..)

➢ Exercice n° 2 (……..…… / 5 points) : QCM Structures de contrôle de base.

① Ecrire un programme qui échange les valeurs de 2 variables a et b, sans affectations multiples. (…. / 1 pt)

aux = a
a = b
b = aux

ou, plus difficile, sans utiliser de variable auxiliaire :
a = a + b
b = a – b
a = a – b

② Entourer le script qui part en boucle infinie (A entier non nul). (D'après sujet 0bis) (…….. / 1 pt)

i = A + 1
while i < A :
  i = i – 1
	i = A + 1
	while i < A :
	  i = i + 1
		i = A – 1
		while i < A :
		  i = i – 1
			i = A – 1
			while i < A :
			  i = i + 1

Une boucle while ne se termine pas lorsque la condition en entête est toujours réalisée !

Pour les boucles 1 et 2, la condition n'est jamais réalisée. Dans la dernière boucle, grâce à l'incrémentation, l'accumulateur i dépassera à un moment A. Pour la 3ème boucle, i décrémente, devient donc de plus en plus petit et ne dépassera jamais A.

③ Le programme suivant n'affiche pas toujours correctement le résultat de x^n. Entourer le couple de valeurs en entrée qui permet de détecter une erreur ?
(D'après sujet 0) (……. / 1 pt)

x ne variant pas, le problème vient de n qui intervient dans le range (n – 1) de l'entête de la boucle.
Range (n – 1) renvoie une liste vide si n – 1 ≤ 0. Donc il faut étudier les cas n = 1 et n = 0.
Si n = 1, on a range(0) donc pas de boucle, res = x pareil que x^1
Si n = 0, on a range(-1) donc pas de boucle, res = x ≠ x^0 = 1 !

```
x = int(input( ))
n = int(input( ))
res = x
for k in range (n – 1) :
    res = res * x
    n = n + 1
print (res)
```

- x =2 et n = 10
- x =2 et n = 2
- x =2 et n = 1
- x =2 et n = 0

④ Le script suivant renverra un résultat équivalent à ? (…….. / 1 pt)

```
p , n = 0 , 3
for k in range (n) :
    for y in range (k) :
        p = p + 1
print(p)
```

1) 1 + 2 + 3
2) 1 + 2
3) 3 × 3
4) 2 × 3

Seule question un peu difficile. On va s'aider d'un tableau d'état des variables.
range(n) = range(3) ↔ [0 , 1 , 2]. Donc k parcourt l'équivalent de cette séquence [0, 1 , 2].

Valeur de k	*range (k) équivaut à*	*nb de tours pour la boucle intérieure*	*Valeur de p*
k =0	*range (0) ↔ [ ] liste vide*	*aucun*	*p = 0*
k = 1	*range (1) ↔ [0]*	*1 tour*	*p = 1*
k = 2	*range (2) ↔ [0 , 1]*	*2 tours*	*p = 2* *p = 3*

Réponse 1) : valable pour n = …. Réponses 3) et 4) valables pour 2 boucles imbriquées indépendantes, l'une de range(3) et l'autre de range (3) pour la réponse 3). Et l'une de range(2) et l'autre de range (3) pour la réponse 4).

⑤ La valeur de « somme » qui s'affichera est donnée mathématiquement par le calcul : (…….. / 1 pt)

```
1  somme=0
2  for k in range(5):
3      if k%2==1:
4          somme=somme+k
5  print(somme)
```

1) 0 + 1 + 2 + 3 + 4
2) 0 + 2 + 4
3) 1 + 3
4) 1 + 3 + 5

Range (5) ↔ (0 , 1 , 2 , 3 , 4). L'instruction ligne 3 teste si k est impair : k s'ajoute alors ligne 4 à l'accumulateur Somme.

⑥ Bonus (……. + 1 pt) (D'après sujet 0) :

Soit le programme suivant. Que peut-on affirmer lorsque le programme se termine (1 seule réponse juste) ?

```
1 capital = int(input( ))
2 interet = int(input( ))
3 montant = capital
4 n = 0
5 while  montant <= 2 * capital :
6         montant = montant + interet
7         n = n+ 1
8 print (n)
```

n = capital / interet
n est le temps qu'il a fallu pour que les intérêts doublent.
capital + n * interet > 2 * capital
capital * n * interet > 2 * capital

Seulement 3 / 21 élèves ont trouvé la bonne réponse en 2019.

*Le programme se termine forcément. En effet, la différence entre montant et 2*capital décroit (car intérêt est positif) et tend vers 0. C'est la théorie du variant de boucle.*

D'après la ligne 6 qui indique une incrémentation du montant des intérêts dans une boucle, montant est un accumulateur.

D'après la ligne 7 qui indique une incrémentation de 1 de n dans une boucle, n est un compteur.

Choix ① : N'importe quoi !

Choix ② : C'est le capital qui doit doubler ! Et non les intérêts !

*Choix ③ : La boucle s'arrête lorsque montant > 2 * capital et le compteur n dira au bout de combien d'étapes n cette condition d'arrêt sera réalisée. Donc il y a n tours de boucle où interet est ajouté à chaque tour à l'accumulateur montant.*

*Donc au bout de n tours, un total de n * interet aura été ajouté à montant, montant valant au départ capital.*

Choix ④ : Montant est un accumulateur, pas un multiplicateur !

➢ Exercice n° 3 (……..…… / 4 points) : Ecriture scientifique. Contrôle1 2016 Tle ISN.

Pour voir plus facilement l'ordre de grandeur d'une quantité, on utilise l'écriture scientifique.

Rappel de 4ème : L'écriture scientifique d'un nombre décimal non nul est de la forme :

$\pm a \times 10^n$ avec $1 \leq a < 10$ et n un entier relatif (a s'appelle la mantisse et n l'exposant).

Exemples : $802{,}3 = 8{,}023 \times 10^2$ $-0{,}0025 = -2{,}5 \times 10^{-3}$ $0{,}04 = \ldots\ldots \times 10^{\ldots\ldots}$ $-3 = \ldots\ldots\ldots \times 10^{\ldots\ldots}$

Concrètement, trouver l'écriture scientifique d'un nombre revient donc à déplacer une virgule d'un ou plusieurs crans, soit à gauche soit à droite puis à compenser ce déplacement de la virgule grâce à la puissance de 10. On veut automatiser cette procédure.

Ce que doit faire votre programme : (…… / 3 pts)

En entrée, le programme doit lire :

- un nombre décimal **« nb » non nul positif** (on se limite au cas des nombres positifs non nuls)**.**

En sortie, votre programme doit afficher la phrase :

- « L'écriture scientifique de « nb » est de la forme « a » × 10 puissance « n ». »
  « nb », la mantisse « a » et l'exposant « n » seront évidemment remplacés automatiquement par leur valeur dans cette phrase. (Rappel : l'instruction print('J\'ai ', a, 'chiens et ', b, ' chats.') affichera « J'ai 3 chiens et 2 chats. » pour a = 3 et b = 2.)

Numérotez les lignes du programme.

Aide : Par quelle opération mathématique se traduit un déplacement de virgule ? Et bien partitionner l'ensemble des cas possibles.

Voir Corrigé Contrôle 1 2017-2018

Très peu réussi en 2019.

Pour s'assurer du bon fonctionnement du programme, il faut tester au moins 5 valeurs, lesquelles ?

(……. / 1 pt)

Il faut tester au moins une valeur correspondant à chacune des 5 parties issues de la partition de l'ensemble des cas possibles : *0 < une valeur < 1 ; 1 ; 1 < une valeur < 10 ; 10 ; 10 < une valeur.*

Question très peu réussie en 2019 ! Pourtant, tester un programme est une étape essentielle de la programmation.

➢ Exercice n° 4 (……..…. / 4 pts) : Conversion Base 10 → Base 2 par divisions successives.

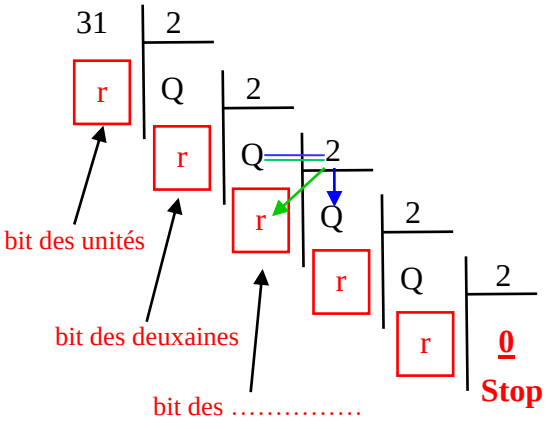
Ecrire en Python un programme qui en entrée reçoit un nombre entier positif ou nul ; et en sortie affichera à l'écran la chaîne de caractères binaires correspondant à ce nombre. Ce programme utilisera l'algorithme des divisions successives.

Exemple : si l'entrée est l'entier 14, le programme affichera la chaîne de caractères '1110'.

Livret Numérisation des nombres p.7 : fait et corrigé en classe !

Très peu réussi en 2019 !

- *Analyse algorithmique : On s'appuie comme d'habitude sur un exemple à la main.*



On effectue une suite de divisions par 2 successives sur les quotients.

D'après le schéma, on voit que : r ← Q % 2 et Q ← Q // 2.

Attention à l'ordre de ces 2 instructions.

Les restes 0 ou 1 seront à accoler à la chaîne résultat par la gauche (du bit de poids le plus faible celui le plus fort).

Et ainsi de suite… → Boucle.

L'algo s'arrête lorsque le quotient est nul → Boucle While.

- *Synthèse Programmation : Autre solution très succincte (d'après sujet 0bis).*

```
1 Nbdépart = int(intpu( ))
2 Quotient = Nbdépart
3 chainebinaire = ' '
4 while Quotient > 0 :
5       chainebinaire = str(Quotient % 2) + chainebinaire  #
6       Quotient = Quotient // 2
7 if Nbdépart = 0 :
8       Print ('0')
9 else :      # Cas général ≠0
10      print(chainebinaire)
```

Initialisation

Corps

Sortie

- *Commentaires :*

Ligne 5 : Calcul du reste dans la division entière (Nbcourant % 2) puis transformation de ce reste (0 ou 1) en caractère (fonction string str( )) puis accolement de ce caractère à gauche de l'accumulateur chaine.

Grâce à cette conversion en caractère du reste, pas besoin de if le reste vaut 0 faire ci, sinon faire ça.

Ligne 6 : Calcul du nouveau quotient en vue du prochain tour de boucle.

Ligne 7 : Le cas particulier de 0 aurait pu être traité dès la ligne 2.

- *Tests : 3 cas en entrée pour Nbdépart à tester : 0 ; 1 et un nb entier quelconque.*