Devoir d'informatique en temps libre n°1 : Corrigé

(1) On obtient les résultats suivants :

Etape	1	2	3	4	5
Dividende	23	11	5	2	1
Quotient	11	5	2	1	0
Reste	1	1	1	0	1

Il faut donc stopper l'algorithme quand le quotient par la division euclidienne par 2 est égal à 0. On lit alors les restes dans l'ordre inverse de leur obtention (une structure de pile est donc naturelle comme nous le verrons).

(2) $141_{10} = 10001101_2$, $858_{10} = 1101011010_2$, $999_{10} = 1111100111_2$. On pouvait en plus vérifier les calculs en s'assurant qu'en base 2 $10001101+1101011010 = 1111100111$ puisque $999 = 858+141$.

(3) En appliquant l'algorithme de la première question on écrit naturellement le programme suivant :

```
def binaire(n) :
  restes = []
  resultat = 0
  while n > 0 :
    restes.append(n\%2)
    n = n//2
  while restes != [] :
    resultat = 10*resultat + restes.pop()
  return resultat
```

Comme on l'a vu, la première partie de l'algorithme donne la liste des restes, dans l'ordre inverse de ce que l'on souhaite écrire. On les stocke donc dans une pile. Pour le calcul du nombre en binaire correspondant, le premier bit lu est celui de poids le plus fort, donc qui doit être le plus à gauche. On décale tous les chiffres d'un cran vers la gauche en multipliant par 10, et on ajoute le bit suivant.

(4) Il suffit de changer les 2 en 3 dans le programme précédent.

($\star$) En adaptant le programme précédent (et en faisant attention que maintenant il ne peut pas renvoyer un entier puisque les caractères de A à F sont utlisés dan sl'écriture, on obtient :

```
def hexadecimal(n) :
  lettres = "ABCDEF"
  restes = []
  resultat = ""
  while n > 0 :
    restes.append(n\%16)
    n = n//16
  while restes != [] :
    bit = restes.pop()
    if bit < 10 :
      resultat = resultat + str(bit)
    else :
      resultat = resultat + lettres[bit-10]
  return resultat
```

L'autre méthode utiise le fait que $16 = 2^4$, donc chaque groupe de 4 bits peut être représenté par un seul caractère en hexadecimal. On effectue donc d'abord la conversion en binaire (en ne renvoyant pas un entier mais une liste d'entier pour chaque bit). Avant d'utiliser la liste des restes, on y ajoute des 0 jusqu'à avoir une liste de longueur multiple de 4. On transforme alors chaque groupe de 4 caractères obtenues en dépilant la liste des restes par le caractère qui lui correspond en hexadécimal (en faisant bien attention au fait que l'on commence par le bit de poids le plus fort), en calculant la valeur de ce groupe. On obtient :

```
def hexadecimal(n) :
    lettres = "ABCDEF"
    restes = []
    binaire = []
    resultat = ""
    while n > 0 :
        restes.append(n\%2)
        n = n//2
    k = len(restes)
    while k\%4 != 0 :
        restes.append(0)
        k = k+1
    while restes != [] :
        bit1 = restes.pop()
        bit2 = restes.pop()
        bit3 = restes.pop()
        bit4 = restes.pop()
        bithexa = 8*bit1 + 4*bit2 + 2*bit3 + bit4
        if bithexa < 10 :
            resultat = resultat + str(bithexa)
        else :
            resultat = resultat + lettres[bithexa-10]
    return resultat
```