Exercice 1 : Compter en base 2 :

Soit le nombre binaire : $n = 1110\ 1111$.

a. Combien de bits composent ce nombre ? 8 bits
b. Donner la valeur en base 2 de $n+1$, $n+2$ et $n+3$

$1111\ 0000$ - $1111\ 0001$ - $1111\ 0010$

Exercice 2 : Compter en base 16 :

Soit le nombre hexadécimal : $n = 1be$. Donner la valeur en base 16 de $n+1$, $n+2$ et $n+3$

$1bf$ - $1c0$ - $1c1$

Exercice 3 : Conversion de la base 2 vers la base 10 :

1- Soit le nombre binaire : $n = 1011$. Quelles est la valeur de ce nombre en base 10 ?

$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11$

2- Soit le nombre binaire : $n = 1111\ 1111\ 1111\ 1111$. Quelles est la valeur de ce nombre en base 10 ? $1 \times 2^{16} - 1 = 65536 - 1 = 65535$

Exercice 4 : Conversion de la base 16 vers la base 10 :

Soit le nombre hexadécimal : $n = 1ff$. Quelles est la valeur de ce nombre en base 10 ?

$1 \times 16^2 + 15 \times 16^1 + 15 \times 16^0 = 256 + 240 + 15 = 511$

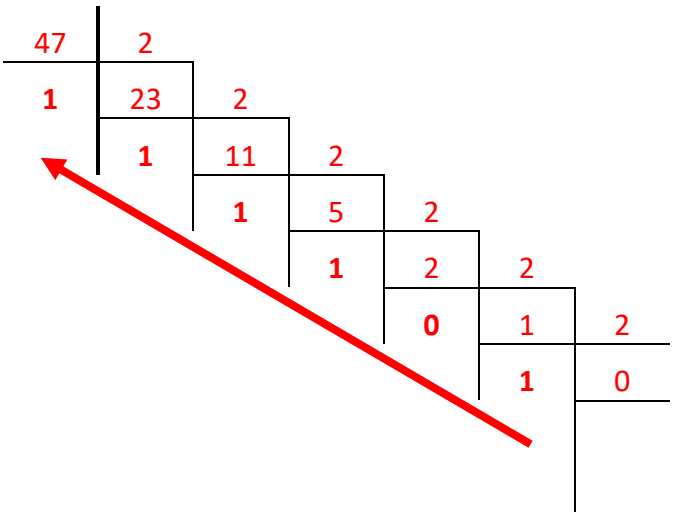
Exercice 5 : Conversion de la base 10 vers la base 2 : ⇨ Convertir $n = 47$ en base 2 en utilisant chacune des 2 méthodes vues en cours :

Méthode 1 : on complète un tableau

32	16	8	4	2	1
1	0	1	1	1	1

47 en base 10 donne ainsi 101111 en base 2

Méthode 2 : restes des divisions euclidiennes par 2



47 en base 10 donne ainsi 101111 en base 2

Exercice 6 : Nombre d'octets :

1- Donner la valeur en hexa du nombre binaire $n = 110\ 0001\ 1101\ 1111\ 0111$

Binaire	Hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

2- L'adresse mémoire ci-dessous est composée de 2 octets donnée ici en hexadécimal. Donner la valeur de ces octets en binaire :

0111 1100 – 0000 0101

Exercice 7 : La fonction *nbr_caractere()* ci-dessous est incomplète. Elle retourne le nombre de caractères d'une chaine de caractère mise en argument :

```
def nbr_caractere(chaine) :
    """
        chaine est une chaine de caractère
        Cette fonction retourne le nombre de caractère de chaine
    """
    n = 0
    for c in chaine :
        n = n + 1
    return n
```

En exécutant les lignes ci-contre dans le *shell*, cette fonction retourne ici 6.

⇨ Compléter le code de cette fonction.

```
>>> n = nbr_caractere("1baf9e")
>>> n
6
```

Exercice 8 : La fonction *affectation()* est définie ci-dessous.

```
def affectation(c) :
    """
        Recoit en argument un chiffre hexadécimal 0 ou 1,2,3,4,
        5,6,7,8,9,a,b,c,d,e,f .
        Retourne la valeur décimale de ce chiffre, soit un entier
        compris entre 0 et 15
    """
    if c == "a" : s = 10
    elif c == "b" : s = 11
    elif c == "c" : s = 12
    elif c == "d" : s = 13
    elif c == "e" : s = 14
    elif c == "f" : s = 15
    else :
        s = int(c)
    return s
```

1- On exécute dans le shell la commande ci-contre `>>> a = affectation("9")`
Quelle est la valeur de la variable *a* après exécution ? a = 9

2- On exécute dans le shell la commande ci-contre `>>> a = affectation("92")`
Quelle est la valeur de la variable *a* après exécution ? a = 92

3- On exécute dans le shell la commande ci-contre `>>> a = affectation("aa")`
Quelle est la valeur de la variable *a* après exécution ? Justifier. on a une erreur d'exécution car le code exécutera l'instruction int(''aa'') qui n'a pas de solution, car la chaine de caractère ''*aa*'' ne peut pas être convertie en nombre entier.

Exercice 9 : La fonction *conversion_hexa()* ci-dessous est incomplète. Elle retourne la conversion en base 10 du nombre mis en argument qui est en base 16.

```
def conversion_hexa(n_hex):
    """
        n_hex est un nombre en base 16, composé de 0,1,2,3,4,
        5,6,7,8,9,a,b,c,d,e,f.
        Retourne la valeur en base 10 de n_hex
    """
    i = nbr_caractere(n_hex) - 1
    n_dec = 0
    for c in n_hex :
        n_dec = n_dec + affectation(c)*16**i
        i = i - 1
    return n_dec
```

En exécutant cette fonction dans le *shell*, on obtient par exemple :

```
>>> a = conversion_hexa("a2f")

>>> a
2607
```

⇨ Compléter le code de cette fonction qui utilisera les fonctions *nbr_caractere()* et *affectation()* définies dans les exercices 7 et 8 précédent.

Exercice 10 : La fonction *couleur()* incomplète ci-dessous prend en argument une couleur définie en hexadécimal (exemple : ''#ab00ff'') . Elle retourne les valeurs r, g, b données en valeur décimale (exemple : 171 , 0 , 255).

```
def couleur(couleur_hexa) :
    """
        Prend en argument une couleur définie sous format
        hexadécimal du type #a1e8fe
        Retourne les valeurs r, g, b en décimal
    """
    r = ""
    g = ""
    b = ""
    i = 0
    for c in couleur_hexa :
        i = i + 1
        if i == 1 : pass
        elif i <= 3 : r = r + c
        elif i <= 5 : g = g + c
        else : b = b + c
    r = conversion_hexa(r)
    g = conversion_hexa(g)
    b = conversion_hexa(b)
    return r,g,b
```

L'exécution de l'instruction `>>> r,g,b = couleur("#ab00ff")` dans le shell retournera les valeurs suivantes pour r, g et b : r = 171 ; g = 0 et b = 255 .

1- Compléter le code de cette fonction qui utilisera la fonction *conversion_hexa()* définie dans l'exercice 9 précédent.

2- Construire un tableau donnant les valeurs prises par chacune des variables de ce code, lorsque la ligne `>>> r,g,b = couleur("#ab00ff")` exécutée :