Exercice 1 : Compter en base 2 :

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

a. Combien de bits composent ce nombre ?

b. Donner la valeur en base 2 de $n + 1$, $n + 2$ et $n + 3$

Exercice 2 : Compter en base 16 :

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

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 ?

2- Soit le nombre binaire : $n = 1111\ 1111\ 1111\ 1111$. Quelles est la valeur de ce nombre en base 10 ?

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 ?

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 :

Méthode 2 :

Exercice 6 : Nombre d'octets :

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

Binaire	Hexa

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

```
   Adresse physique . . . . . . . . . . . . : 7C-05-
```

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
    """
```

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 ?

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

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

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
    """
```

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
    """
```

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 :

couleur_hexa	r	g	b		
'#ab00ff'					

Exercice 11 :

En exécutant le script incomplet ci-dessous, on obtient dans le shell :

```
>>> (executing file "ex_tictac.py")
tic tac tic tac tic tac tic tac tic tac
```

⇨ Compléter ce script afin de pouvoir obtenir le même fonctionnement :

```
# Définition des fonctions
def



# Programme principal
tictac(10)
```