Exercice n°1 : conversion binaire -> décimal

Exercice 1: Ecrire une fonction **conversion** qui réalise la conversion de binaire à décimal sur un octet. La fonction prend pour paramètre un nombre binaire de 8 chiffres. Ce binaire sera écrit entre guillemets (un string).

```
1. def conversion(b):
2.     return b[0]*128 + ..
3.
4. # Exemple d'utilisation
5. >>> conversion(“10001000")
6. 136
```

Le programme utilisera l'indice des positions de chaque chiffre:
`conversion = b[0]*128 + b[1]*64 + …`

Exercice n°2 : inverser les bits

Ecrire une fonction **permute** qui permute tous les bits d'un nombre binaire sur 1 octet. La fonction prend pour paramètre `nombre` et retourne une chaine de 8 bits, tous différents des bits de `nombre`. Il s'agit du complément à 1.

Le principe est le suivant:
* On créé une chaine de caractères `nombre2` vide, pour stocker les nombres permutés.
* On parcourt les chiffres `c` de `nombre` un à un, dans l'ordre.
* Si `c` est un "1", on ajoute "0" à `nombre2` de la manière suivante:

`nombre2 = nombre2 + "1"`

* Sinon, `c` est un "0" donc on ajoute "1"

```
1. def permute(nombre):
2.     nombre2 = ..
3.     for c in ..       :
4.         if c == .. :
5.             ..
6.         else:
7.             ..
8.     return ..
9. # Exemple d'utilisation
10. >>> permute(“10001000")
11. “ .. .. .. .. ..”
```

Exercice n°3 :Retourner la chaine de caractères

compléter le script de la fonction **renverse** qui prend pour paramètre `mot` et qui retourne le `mot` à l'envers.

Le principe est le suivant:

* On créé une chaine de caractères `mot2` vide, pour stocker les caractères renversés.
* On parcourt les caractères de mot un à un, dans l'ordre.
* On ajoute chaque nouveau caractère à mot2

```
1. def renverse(mot):
2.     mot2 = ..
3.     for c in mot:
4.         mot2 = ..
5.     return ..
6.
7. Exemple:
8. >>> renverse("LEON")
9. NOEL
```

Exercice n°4: Addition binaire

Programmer la fonction `ajoute1` qui ajoute 1 au `nombre` binaire sur 1 octet. Cette fonction aura pour paramètre `nombre`, une chaine de caractères avec 8 bits. Créer une chaine vide `nombre2.` Utiliser la fonction `renverse` pour parcourir les chiffres de `nombre` à l'envers. Commencer l'addition sur le chiffre le plus à gauche de nombre. A chaque itération, ajouter `retenue` au chiffre `c`, et l'ajouter à la chaine `nombre2.`

`Retourner alors la chaine nombre2, mais retournée.`

```
>>> ajoute1("11010111")
"11011000"
>>> ajoute1("11111111")
"00000000"
```

```
1. def ajoute1(nombre):
2.     nombre2 = ..
3.     retenue = 1
4.     for c in renverse(.. :
5.         c = int(c) + retenue
6.         if c <= 1:
7.             retenue = ..
8.             nombre2 = nombre2 + str(..
9.         else:
10.            retenue = ..
11.            c = 0
12.            nombre2 = nombre2 + str(..
13.    return renverse(..
```

Prolongement: Vous pouvez à présent écrire une fonction qui retourne l'opposé d'un entier positif inférieur ou égal à 128, pour obtenir son négatif (méthode du complément à 2).

Exercice n°5 : nombre d'occurrences

Programmer la fonction occurrences qui prend `mot` ainsi que `caracteres` comme paramètres. Cette fonction retourne le nombre fois où le caractère est présent dans le mot:

```
>>> occurrences("lustucru","u")
3
>>> occurrences("lustucru","y")
```

```
1. def occurrences(mot, caractere):
2.     s = 0
3.     for c in ..    :
4.         if c ==          :
5.             s =
6.     return ..
```

Exercice a : Compter les voyelles dans une chaîne

Écrire une fonction qui compte le nombre de voyelles (a, e, i, o, u, y) dans une chaîne donnée.

Exemple :

```
>>> compter_les_voyelles("Python est génial!"):
5
```

Aide: utiliser la fonction occurrences.

Exercice b : Vérifier si une chaîne est un palindrome

Écrire un programme qui vérifie si une chaîne est un palindrome (même lue de gauche à droite et de droite à gauche).

Exercice c : Extraire les sous-chaînes

Écrire une fonction qui extrait une sous-chaîne à partir d'un indice de début et d'un indice de fin donnés, et qui l'affiche.

Exemple :

```
>>> extraction("Hello World", 6, 11)
"World"
```

Indice : *Utilise la syntaxe de découpe de chaînes (slicing) :* `chaine[start:end]`

Exercice d : Supprimer les espaces d'une chaîne

Écrire une fonction qui supprime tous les espaces d'une chaîne, sans utiliser la méthode `split`

Exemple :

```
>>> supprimer_espaces("Bonjour tout le monde")
"Bonjourtoutlemonde"
```

Indice : *Utilise une boucle* `for` *et ajoute les caractères non espacés à une nouvelle chaîne.*

CORRECTIONS

Exercice 1

```
1. def conversion(b):
2.     return b[0]*128 + b[1]*64 + b[2]*32 + b[3]*16 + b[4]*8 +
3.            b[5] * 4 + b[6] * 2 + b[7]
4. # Exemple d'utilisation
5. >>> conversion(“10001000")
6. 136
```

Exercice 2

```
1. def permute(nombre):
2.     nombre2 = “”
3.     for c in nombre      :
4.         if c == “1” :
5.             nombre2 = nombre2 + “0”
6.         else:
7.             nombre2 = nombre2 + “1”
8.     return nombre2
9. # Exemple d'utilisation
10. >>> permute(“10001000")
11. “ .. .. .. .. ..”
```

Exercice 3

```
1. def renverse(mot):
2.     mot2 = “”
3.     for c in mot:
4.         mot2 = c + mot2
5.     return mot2
6.
7. Exemple:
8. >>> renverse("LEON")
9. NOEL
```

Exercice 4

```
1. def ajoute1(nombre):
2.     nombre2 = ""
3.     retenue = 1
4.     for c in renverse(nombre) :
5.         c = int(c) + retenue
6.         if c <= 1:
7.             retenue = 0
8.             nombre2 = nombre2 + str(c)
9.         else:
10.            retenue = 1
11.            c = 0
12.            nombre2 = nombre2 + str(c)
13.    return renverse(nombre2)
```

```
1. >>> ajoute1("11010111")
2. "11011000"
3. >>> ajoute1("11111111")
4. "00000000"
```

Exercice 5

```
1. def occurrences(mot, caractere):
2.     s = 0
3.     for c in mot   :
4.         if c == caractere:
5.             s = s + 1
6.     return s


>>> occurrences(“lustucru”,”u”)
3
>>> occurrences(“lustucru”,”y”)
```