Python – Changement de base

Les bases des bases

Nous allons faire des fonctions permettant de convertir un nombre décimal en une autre base ou d'une autre base en décimal.
Les nombres en base 10 seront représentés par des nombres entiers et ceux des autres bases seront représentés par des chaînes de caractères.
Pour pouvoir gérer les bases de plus de 10 chiffres, nous allons rajouter un alphabet :

```
alphabet = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Ainsi au moment de rajouter le chiffre `c` au résultat, nous pourrons rajouter `alphabet[c]`.

```
>>> alphabet[5]
'5'
>>> alphabet[10]
'A'
>>> alphabet[15]
'F'
```

Pour retrouver la "valeur" d'un chiffre d'un nombre en base b, il est possible d'utiliser la commande suivante :

```
>>> alphabet.index("5")
5
>>> alphabet.index("A")
10
>>> alphabet.index("F")
15
```

On rappelle qu'il est possible d'ajouter des caractères en début ou en fin de chaînes à l'aide des commandes suivantes :

```
>>> resultat = "11"
>>> resultat = "0" + resultat
>>> resultat
'011'
>>> resultat += "2"
>>> resultat
'0112'
```

On rappelle aussi que le quotient de la division euclidienne est donné par `a // b` et le reste par `a % b`.
Pour parcourir un texte, on peut le faire lettre par lettre :

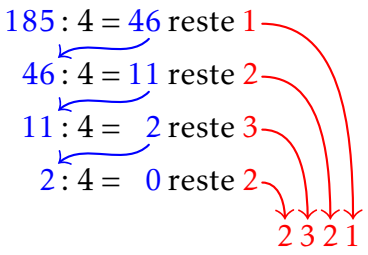
```
for lettre in texte:
    print(lettre)
```

Ou alors, indice par indice :

```
for indice in range(len(texte)):
    print(texte[indice])
```

De la base 10 à la base b

Exercice 1 : Compléter la fonction `conversion(nb, b)` qui prend un entier `nb` en base 10 et renvoie son écriture en base b. Cette fonction utilise la méthode des divisions successives par b. Attention, il faut rajouter les restes de droite à gauche. Pour rappel la division entière s'écrit `n // b` et le reste s'obtient en faisant `n % b`.



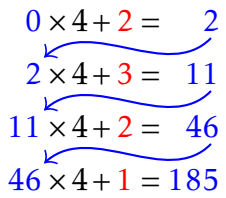

```
def conversion(nb, b):
    res = ""
    while ...: # tant qu'il reste plus d'un chiffre à mettre
        res = alphabet[...] + res # on rajoute à gauche
        nb = ...
    res = ... # on rajoute le dernier chiffre
    return res
```

```
>>> conversion(1, 2)
'1'
>>> conversion(10, 2)
'1010'
>>> conversion(255, 2)
'11111111'
```

```
>>> conversion(255, 4)
'3333'
>>> conversion(255, 6)
'1103'
>>> conversion(255, 16)
'FF'
```

De la base b à la base 10

Exercice 2 : Compléter la fonction `deconversion(nb, b)` qui prend un nombre `nb` en base `b` et renvoie sa valeur en base 10. Nous allons utiliser la méthode de Horner.



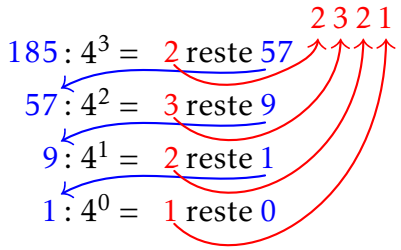

```
def deconversion(nb, b):
    res = 0
    for chiffre in nb:
        res = res * ... + ...
    return res
```

```
>>> deconversion("1001", 2)
9
>>> deconversion("1011", 2)
11
>>> deconversion("1010", 2)
10
```

```
>>> deconversion("10101010", 2)
170
>>> deconversion("7F", 16)
127
>>> deconversion("55", 6)
35
```

Les autres méthodes

Exercice 3 : Compléter la fonction `conversion2(nb, b)` qui implémente la méthode des divisions par les puissances décroissantes de `b` pour convertir un nombre de la base 10 à la base `b`. Pour cela, on commence par chercher la plus grande puissance de `b` qui est inférieure ou égale à `nb` et ensuite on fait les divisions successives par les puissances, de plus en plus petites.




```
def conversion2(nb, b):
    puiss = ... # la plus petite des puissances de b
    while ...: # on cherche la plus grande puissance de b inférieure ou égale à nb
        puiss = ... # on passe à la puissance suivante
    res = ""
    while ...: # tant qu'on doit diviser
        chiffre = ... # la valeur du chiffre à ajouter au résultat
        res = ... # on l'ajoute du bon côté
        nb = ... # on met à jour le nombre qu'il faut convertir
        puiss = ... # on diminue la puissance
    return res
```

Vérifier qu'elle donne bien les mêmes valeurs que `conversion(nb, b)`.

Exercice 4 : Compléter fonction `deconversion2(nb, b)` qui implémente la méthode par la multiplication par des puissances de `b` en allant de droite à gauche pour convertir un nombre de la base b en décimal. Par exemple :

$$2321_4 = 1 \times 4^0 + 2 \times 4^1 + 3 \times 4^2 + 2 \times 4^3 = 185$$

Il faut parcourir les positions de droite à gauche. la position la plus à droite est `len(nb)-1` et celle la plus à gauche est `0`.

```
def deconversion2(nb, b):
    res = ...
    puiss = ...
    for i in range(..., ..., ...):
        chiffre = alphabet.index(nb[i])
        res = res + ...
        puiss = ...
    return res
```

Vérifier qu'elle donne bien les mêmes valeurs que `deconversion(nb, b)`.

Pour aller plus loin

Exercice 5 : Modifier au moins une des fonctions de conversion en rajoutant un paramètre : `conversion(nb, b, k=0)`. Si `k` = 0, le comportement est le même. Si `k` > 0, alors il faut produire un nombre de `k` chiffres, soit en tronquant, soit en rajoutant des 0 à gauche.
Vous pouvez utiliser les commandes suivantes :

```
>>> "111".rjust(5, "0") # 5 chiffres au moins, avec autant de 0 à gauche que nécessaire
'00111'
>>> "111".rjust(7, "0") # 7 chiffres au moins, avec autant de 0 à gauche que nécessaire
'0000111'
>>> "111".rjust(2, "0") # 5 chiffres au total, avec autant de 0 à gauche que nécessaire
'111'
>>> "1234"[-2:] # On garde les 2 derniers symboles
'34'
>>> "1234"[-3:] # On garde les 3 derniers symboles
'234'
>>> "1234"[-5:] # On garde les 5 derniers symboles
'1234'
```

Voici quelques exemples d'utilisation :

```
>>> conversion(21, 2)
'10101'
>>> conversion(21, 2, 8)
'00010101'
>>> conversion(21, 2, 3)
'101'
```