Lycée Jean Moulin **Draguignan**
NSI 1re Année .....

Python – Changement de base

Nom :

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`.

De la base 10 à la base b

Il s'agit de réaliser l'algorithme suivant:

Données : nb : entier naturel
b : entier naturel (0 < b <=16)
mot ← une chaîne de caractère vide
Tant que $nb > 0$ **faire**
    mot ←alphabet[nb%b] + mot
    nb ← nb//b
fin tant que
retourner mot

17	2	8	2	4	2	2	2	1	2
1	8	↗0	4	↗0	2	↗0	1	↗1	0

- au 1er tour: mot = "1" + mot = "1" (nb = 8)
- au second tour: mot = "0" + mot = "01" (nb = 4)
- au 3ème tour: mot = "0" + mot = "001" (nb = 2)
- au 4ème tour: mot = "0" + mot = "0001" (nb = 1)
- au dernier tour: mot = "1" + mot = "10001" (nb = 0)

? Exercice 1 :

Écrire une fonction `conversion(nb,b)` qui prend un entier `nb` en base 10 et retourne son écriture en base *b*. En suivant l'algorithme précédent.
Voici quelques exemples d'utilisation :

```
>>> conversion(1, 2)
'1'
>>> conversion(10, 2)
'1010'
>>> conversion(255, 2)
'11111111'
>>> conversion(255, 4)
'3333'
>>> conversion(255, 6)
'1103'
>>> conversion(255, 16)
'FF'
```

? Exercice 2 :

Quelle est la valeur retournée par `conversion(0,2)` ? Corriger votre fonction si nécessaire.

De la base b à la base 10

Il s'agit de réaliser l'algorithme suivant:

Données : nb : chaîne de caractère
b : entier naturel (0 < b <=16)
res ← 0
n ← longueur de nb - 1
Pour chaque lettre in nb **faire**
    res = res + alphabet.index(lettre)*b**(n)
    n = n - 1
retourner res

"10001" →
$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 16 + 1$
$= 17$

Exercice 3 :

Écrire une fonction `deconversion(nb,b)` qui prend un nombre `nb` en base `b` (sous forme d'une chaîne de caractères) et retourne sa valeur en base 10. En utilisant l'algorithme précédent.
Voici quelques exemples d'utilisation.

```
>>> deconversion("1001", 2)
9
>>> deconversion("1011", 2)
11
>>> deconversion("1010", 2)
10
>>> deconversion("10101010", 2)
170
>>> deconversion("7F", 16)
127
>>> deconversion("55", 6)
35
```

Pour aller plus loin...

Exercice 1 : Modifier la fonction 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")
'00111'
>>> "111".rjust(7, "0")
'0000111'
>>> "111".rjust(2, "0")
'111'
>>> "1234"[-2:]
'34'
>>> "1234"[-3:]
'234'
>>> "1234"[-5:]
'1234'
```

Voici quelques exemples d'utilisation :

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