Changement de base - Récursivité

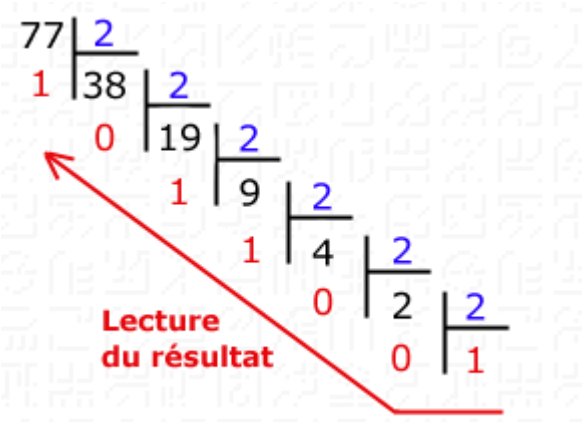
Quelques rappels...

Du décimal vers une base $2 \leq b \leq 16$

Comment convertir 77 en base 2?

On rappelle la méthode ci-contre, le résultat est 1001101.

C'est à dire le dernier quotient non nul suivi des restes en commençant par le dernier.



Voici un programme itératif qui réalise ce changement de base

```
def decTob(n,b):
    assert (b>1 and b<17) ,"b doit être compris entre 2 et 16"
    signes=["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"]
    mot = ""
    while n!= 0:
        mot = signes[n%b] + mot
        n = n//b
    return mot
```

Analysons ce programme

• La première ligne permet de s'assurer que les conditions sur b sont assurées.

• La liste `signes` nous permet d'avoir accès aux symboles représentant les nombres jusqu'à la base 16.

• On utilisera la variable `mot` de type **`str`** pour le résultat.

• Tant que `n! = 0` (tant que le quotient n'est pas nul), on ajoute par la gauche le reste au résultat et on remplace n par le nouveau quotient `n//b`

Exercice 1 :

Tester ce programme avec différentes bases
Pour vérifier:

```
print(bin(77)[2:]) ## affiche 77 en base 2
print(oct(77)[2:]) ## affiche 77 en base 8
print(hex(77)[2:]) ## affiche 77 en base 16
```

Une version récursive...

l'objectif de ce TP est d'écrire une version récursive de ce programme.
L'idée est : `decTobr(n,b)`=decTobr(n//b,b) + reste

Rappelons les trois règles :

• Un algorithme récursif doit avoir un "état trivial" , cela permet d'avoir une condition d'arrêt.

• Un algorithme récursif doit conduire vers cet "état d'arrêt", cela permet de ne pas faire une infinité d'appels récursifs.

• Un algorithme récursif s'appelle lui même...

Question 1:

Dans notre cas quel est "l'état trivial"?

.....
.....

Question 2:

Expliquer ce qui va conduire à cet "état trivial".

.....
.....
.....
.....
.....
.....
.....

Exercice 2 :

Réaliser une version récursive du programme précédent.

```
# Écrire votre programme ici
```