I Conversion en binaire

Pour mémoire, rappelons l'algorithme classique pour trouver les bits de la décomposition binaire sur un exemple. On cherche la décomposition de 35

$$\begin{array}{r|l} 35 & 2 \\ \hline 15 & 17 \\ 1 & \end{array}$$

Notons que le reste vaut 1 et le quotient 17. On sait alors que le dernier bit est 1 et on recommence avec le quotient obtenu.

$$\begin{array}{r|l|r|l|r|l|r|l|r|l} 17 & 2 & 8 & 2 & 4 & 2 & 2 & 2 & 1 & 2 \\ \hline 1 & 8, & 0 & 4, & 0 & 2, & 0 & 1, & 1 & 0 \end{array}$$

On a répété l'opération sur tous les quotients successif jusqu'à obtenir 0 comme quotient.
Ainsi, $35 = 100011_{(2)}$.

II Implémentation en python

Remarquons que l'on souhaite effectuer la même opération (division par 2, noter le reste) sur une suite de nombre qui commence par n (le nombre dont on cherche la décomposition binaire) et on s'arrête quand ce nombre vaut 0.
Ainsi nous allons utiliser une boucle while qui modifie la valeur d'une variable dont la valeur initiale est n et qui se poursuis tant que cette variable est non nulle.
Dans cette implémentation, on se contente de construire la liste des bits, le premier étant le bit de poids faible (on conserve l'ordre de calcul, qui n'est pas l'ordre de lecture)

```python
def chiffres_binaires(n):
    q = n # première valeur du quotient
    L = [] # la liste des bits déjà calculés
    while q != 0:
        bit = q % 2  # prend comme valeur la suite des bits de n
        L.append(bit)  # on a trouvé un nouveau bit, stockons le
        q = q // 2
    return L
```

III Chiffres en base 10

On va adapter les notions précédentes à la base 10 pour répondre à la question suivante :
Créer une fonction `palindrome(n)` qui prend comme argument un entier naturel n et retourne le booléen indiquant si n est un palindrome ie peut se lire de gauche à droite comme de droite à gauche.
Avant de tourner la page, prenez un temps pour tenter de répondre à cette question, en 3 temps : créer la liste des chiffres décimaux de n, tester si une liste représente un palindrome puis combiner les deux fonctions.

```python
def liste_chiffres(n):
    L = []
    q = n
    while q != 0:
        L.append(q % 10)
        q = q // 10
    # liste des chiffres, à l'envers par rapport à la lecture
    return L

def est_palindrome_list(L):
    dernier = len(L) - 1
    # on fait trop de test, mais peu importe.
    for i in range(dernier + 1):
        if L[i] != L[dernier - i]:
            return False
    return True

def est_palindrome(n):
    # testons si la liste des chiffres de n (à l'envers) est égale à sa renversée.
    return est_palindrome_list(liste_chiffres(n))
```