TP du 3 octobre

1 Renverser une liste

Après chaque ligne recopiée, vous appuyerez sur la touche "Entrée" (anciennement, du temps de nos grands-parents, « retour chariot »).
Dans la console, taper

```
>>> L = [7,8,9]
>>> L.reverse()
```

Que se passe-t-il ?
Taper alors

```
>>> L
```

Quel est l'effet de la commande `L.reverse()` ?

Dans l'éditeur de texte, créer une fonction `MaFonctionReverse(L)` qui prend en argument une liste `L` et renvoie la liste « dans l'autre sens ».

2 Schéma de Hörner

Soit $b \geqslant 2$ un entier.

• Pour connaître la valeur décimale de l'entier $[c_s \dots c_0]_b$ écrit en base b, il s'agit de calculer :

$$c_s b^s + c_{s-1} b^{s-1} + \cdots + c_1 b + c_0.$$

Combien d'opérations comporte ce calcul ? .....

• Écrire une fonction `EvaluationNaive(L, b)` qui prend en argument une liste d'entiers $L = [c_0, \dots, c_s]$ et un entier $b \geqslant 2$ et qui retourne $\sum_{k=0}^{s} c_k b^k$ le plus « naïvement » possible.

• On se propose maintenant de calculer la quantité suivante (merci Hörner)

$$\Big(\big((c_s b + c_{s-1})b + c_{s-2}\big)b + \cdots + c_1\Big)b + c_0.$$

Combien d'opérations comporte ce calcul ? .....

• Compléter cet algorithme de Hörner écrit en pseudo code.

Entrée : L = liste contenant les chiffres de n en base $b \geqslant 2$.
Sortie : Calcul de $n = c_s b^s + c_{s-1} b^{s-1} + \cdots + c_1 b + c_0$ via Hörner
  $n = 0$
  pour c parcourant $L =$ ..... **faire**

    ...

  fin pour
  renvoie n

• Écrire une fonction `Horner(L,b)` qui fait ce qu'il faut !

3 Conversion en base b

Écrire en pseudo code l'algorithme permettant d'écrire un entier $n \in \mathbb{N}^*$ en base b.
On stockera les chiffres dans une liste.
Que renvoie votre fonction si on l'applique à $n = 1987$ et $b = 10$? Renvoie-t-elle [1,9,8,7] ou [7,8,9,1] ?
Peu importe la réponse, mais vous devez être conscient de ce que retourne **votre** fonction.

Écrire en Python une fonction `Conversion(n,b)` qui prend en argument un entier $n \in \mathbb{N}^*$ et un entier $b \geqslant 2$ et qui renvoie l'écriture de n en base b.
Attention, voici des consignes

- On donnera le résultat sous forme d'une liste.
- On utilisera uniquement `L.append()` pour une liste `L` (autrement dit, interdiction d'utiliser une commande du type `L = L [truc]+`).
- On pourra faire appel à des fonctions déjà codées dans ce TP.

Votre fonction devra avoir l'effet suivant :

```
>>> Conversion(1987, 10)
[1,9,8,7]
```

Recopier votre code ici pour l'apprendre.

Avec la fonction créée, reprendre les questions du cours : convertir 2024 en binaire, en hexadécimale, en base 7.

4 Écriture sur k bits

Dans la console, écrire, en une ligne, une commande permettant d'écrire la liste `[0,0,0,....,0]` ayant 33 zéros. M'appeler une fois ceci fait.

Écrire une fonction `Binaire_NombreBitsFixe(n,k)` qui prend en argument deux entiers naturels n et k avec $0 \leqslant n < 2^k$ et renvoie une liste donnant l'écriture de n sur k bits.

```
>>> Binaire_NombreBitsFixe(5,7)
[0,0,0,0,1,0,1]
```

Règles du jeu. On pourra utiliser la concaténation de listes via le symbole + et s'inspirer de la question préliminaire.
Bonus. Pour sécuriser la fonction, on peut ajouter une ligne du type `assert n >= 0 and n < 2**k`

5 Chaîne de caractères

Que fait la fonction suivante ? Quel est le type de la variable renvoyée ?

```
def mystere(n,b):
    S = ''
    q = n
    while q != 0:
        r, q = q%b, q//b
        S = str(r) + S
    return S
```

6 La suite de Conway

La suite de Conway est une suite inventée en 1986 par le mathématicien John Horton Conway, initialement sous le nom de « suite audioactive ».
Elle est également connue sous le nom anglais de « Look and Say » !
Dans cette suite, un terme se détermine en annonçant les chiffres formant le terme précédent, le terme initial étant 1.

```
1
1 1
2 1
1 2 1 1
1 1 1 2 2 1
```

Par exemple, la 4ème ligne 1211 dit que la 3ème ligne est constituée d'*un* nombre 2 suivi d'*un* nombre 1.
Quelle est la somme des nombres de la 60ème ligne ?
Indication : on pourra d'abord créer une fonction `ConwayTransformation` qui produit :

```
>>> L = [x for x in 'aaabbaa']
>>> L
['a', 'a', 'a', 'b', 'b', 'a', 'a']
>>> ConwayTransformation(L)
[3, 'a', 2, 'b', 2, 'a']
```

TP du 3 octobre

corrigés

2

L'algorithme naïf d'évaluation comporte s additions, puis $1+2+\cdots+s$ multiplications, c'est-à-dire $s+\dfrac{s(s+1)}{2}$ opérations : on dit que la complexité est quadratique (de l'ordre de s^2).

L'algorithme de Hörner comporte 2 opérations par parenthèse. Il y a $s-1$ parenthèses ici :

$$\Big(\big((c_s b + c_{s-1})b + c_{s-2}\big)b + \cdots + c_1\Big)b + c_0.$$

Il faut imaginer une big parenthèse englobant le tout, d'où s parenthèses.
Bilan : il y a $2s$ opérations dans Hörner.

Entrée : $L = [c_s, \ldots, c_0]$ et $b \geqslant 2$
Sortie : Calcul de $n = c_s b^s + c_{s-1} b^{s-1} + \cdots + c_1 b + c_0$ via Hörner
  s = longueur de $L - 1$
  $n = L[0]$
  pour $i = 1$ **à** s **faire**
    $n = n * b + L[i]$
  fin pour
  renvoie n

```
25 def Horner(L,b):
26     s = len(L)-1
27     n = L[0]
28     for i in range(1,s+1):
29         n = n*b + L[i]
30     return n
31
32
33 def HornerBis(L,b):
34     s = len(L)-1
35     n = L[0]
36     for c in L[1:]:
37         n = n*b + c
38     return n
```

6

Je mets ici toutes les variantes que j'ai écrites.

```
39 def ConwayTransformation(L):
40     # Pour L = [x for x in 'aaabbaa'] renvoie S = [3,'a', 2,'b', 2,'a']
41     c = 0
42     a = L[0]
43     S = [c, a]
44     for e in L:
45         if e == S[-1]:
46             S[-2] += 1
47         else:
48             S += [1,e]
49     return S
50 
51 def ConwayTransformationOne(L):
52     S = []
53     c = 0
54     a = L[0]
55     for e in L:
56         if e == a:
57             c += 1
58         else:
59             S += [c,a]
60             c = 1
61             a = e
62     S += [c,a]
63     return S
64 
65 def ConwayTransformationBis(L):
66     S = []
67     j = 0
68     while j < len(L):
69         c = 0
70         while j+c < len(L) and L[j+c] == L[j]:
71             c += 1
72         S += [c, L[j]]
73         j += c
74     return S
75 
76 def ConwayTransformationTer(L):
77     S = []
78     i = 0
79     while i < len(L):
80         e = L[i]
81         c = 1
82         i += 1
83         while i < len(L) and L[i] == e:
84             c += 1
85             i += 1
86         S += [c,e]
87     return S
88 
89 def ConwayTransformationFM(L):
90     n = 1
91     res = []
92     for i in range(len(L)):
93         if i == len(L)-1 or L[i] != L[i+1]:
94             res += [n,L[i]]
95             n = 0
96         n += 1
97     return res
```


```
98
99  def ConwayTransformationFMbis(L):
100     n = 1
101     res = []
102     for i in range(len(L) - 1):
103         if L[i] != L[i+1]:
104             res += [n,L[i]]
105             n = 0
106         n += 1
107     return res + [n,L[-1]]
108
109
110 def DessinConway(n):
111     L = [1]
112     print(' '.join([str(elt) for elt in L]))
113     for _ in range(2,n+1):
114         L =  ConwayTransformation(L)
115         print(' '.join([str(elt) for elt in L]))
116
117
118 def ConwaySum(n):
119     L = [1]
120     for _ in range(2,n+1):
121         L =  ConwayTransformation(L)
122     return len(L)
123
124
125 # import time
126 # start_time = time.time()
127 # ConwaySum(60)
128 # print("--- %s seconds ---" % (time.time() - start_time))
```