TP du 17 octobre

1 **Rappel sur le schéma de Hörner — à faire sur le papier —**
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 ? .....

Le nombre d'opérations est de l'ordre de .....

• 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

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

Combien vaut-elle (merci Hörner) ? .....

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

Le nombre d'opérations est de l'ordre de .....

• Écrire une fonction `Horner(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 $\sum_{k=0}^{s} c_k b^k$ en suivant le schéma de Hörner.

2 Rappel sur la conversion en base b — à faire sur le papier —

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 une liste donnant les chiffres de l'écriture de n en base b.

Règle du jeu.

— 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, ou bien faire comme si on les avait déjà codées.

— Votre fonction devra avoir l'effet suivant :

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

3 La transformation de Conway

Écrire une fonction `ConwayTransformation(L)` qui prend en argument une liste `L` de chaînes de caractères de longueur 1 et qui a l'effet suivant :

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

Règle du jeu. On n'utilisera pas de while. Uniquement des for et des if.
Idéalement pas plus de deux indentations (autrement dit, 1 for et 1 if, ou bien ...).

4 La suite de Syracuse

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$\begin{cases} u_0 \in \mathbb{N}^* \\ \forall\, n \in \mathbb{N},\ u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases} \end{cases}$$

La conjecture de Syracuse affirme que la suite u finit toujours par atteindre 1.
Les termes de la suite ont été imagés par les mathématiciens : ils ont emprunté le vocabulaire de l'aviation.
Un « vol » est une suite u.

— Le vol numéro $a \in \mathbb{N}^*$ est la suite de premier terme a.
— L'altitude maximale du vol est la plus grande altitude rencontrée pendant le vol.
— La durée du vol est le temps nécessaire à atteindre pour la première fois la hauteur 1 : c'est le plus petit entier n tel que $u_n = 1$.

1. Écrire une fonction `suivant(u)` qui prend en argument un terme `u` de la suite et renvoie le suivant.
2. Écrire une fonction `duree_vol` qui prend en argument un entier a et qui renvoie la durée du vol numéro a.
   On doit avoir

```
>>> duree_vol(11)
14
```

3. Écrire une fonction `altitude_max_vol` qui prend en argument un entier a et qui renvoie l'altitude maximal du vol numéro a.

```
>>> altitude_max_vol(11)
52
```

4. Écrire une fonction `parcours_vol` qui prend en argument un entier a qui renvoie la liste de toutes les altitudes parcourues pendant le vol numéro a (autrement dit qui renvoie $u_0, u_1, \ldots, u_n$ où n est tel que $u_n = 1$).
5. Écrire une fonction `dessin_vol` qui prend en argument un entier a et qui renvoie le dessin du vol numéro a.
6. Écrire une fonction `plus_long_vol` qui renvoie le numéro du vol, compris entre 3 et 10^4, ayant le plus long vol.
   Quelle est la durée de ce vol? Son altitude maximale?

5 Le triangle de Pascal

1. Écrire une fonction `Pascal` qui prend en argument $n \in \mathbb{N}$ et retourne les $n+1$ premières lignes du triangle de Pascal dans une liste de listes.
   Règle du jeu. On utilisera uniquement la formule du triangle de Pascal (pas de factorielle donc!).

   Par exemple, on doit avoir

```
>>> Pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

2. Écrire une fonction `DessinTriangle` qui prend en argument $n \in \mathbb{N}$, qui ne renvoie rien (cela s'appelle une *procédure*, mais qui dessine les $n+1$ premières lignes du triangle de Pascal (on utilisera évidemment la fonction précédente).

```
>>> DessinTriangle(4)
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

TP du 17 octobre

corrigés

1

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

```
def Horner(L,b):
    s = len(L)-1
    n = L[0]
    for i in range(1,s+1):
        n = n*b + L[i]
    return n


def HornerBis(L,b):
    s = len(L)-1
    n = L[0]
    for c in L[1:]:
        n = n*b + c
    return n
```


3

```
def ConwayTransformation(L):
    n = 1
    res = []
    for i in range(len(L)):
        if i == len(L)-1 or L[i] != L[i+1]:
            res += [n,L[i]]
            n = 0
        n += 1
    return res

def ConwayTransformationBis(L):
    n = 1
    res = []
    for i in range(len(L) - 1):
        if L[i] != L[i+1]:
            res += [n,L[i]]
            n = 0
        n += 1
    return res + [n,L[-1]]
```


4

```
def suivant(u):
    if u%2 == 0:
        return u//2
    else:
        return 3*u+1

def duree_vol(a):
    n = 0
    u = a
    while u != 1:
        n = n+1
        u = suivant(u)
    return n+1

def altitude_vol(a):
    alt = a
    u = a
    while u != 1:
        u = suivant(u)
        alt = max(alt, u)
    return alt


def parcours_vol(a):
    H = [a]
    u = a
    while u != 1:
        u = suivant(u)
        H.append(u)
    return H


def plus_long_vol(N = 10**4):
    duree_max = 1
    a_max = 1
    for a in range(3, N+1):
        d = duree_vol(a)
        if d > duree_max:
            duree_max = d
            a_max = a
    return a_max, duree_max, altitude_vol(a_max)


################################### Dessinons
import matplotlib.pyplot as plt

def dessin_vol(a):
    H = parcours_vol(a)
    T = list(range(len(H)))
    plt.plot(T,H)
    plt.show()
```


5

```
def Pascal(n):
    T = []
    LigneZero = [1]
    T.append(LigneZero)
    for i in range(1,n+1):
        Ligne_i = [1] + [T[-1][k-1] + T[-1][k] for k in range(1,i)] + [1]
        assert len(Ligne_i) == i+1
        T.append(Ligne_i)
    return T

def DessinTriangle(n):
    T = Pascal(n)
    for ligne in T:
        s = ''
        for elt in ligne:
            s += str(elt) + " "
        print(s)

def DessinTriangle(n):
    T = Pascal(n)
    for ligne in T:
        print(' '.join([str(elt) for elt in ligne]))
```