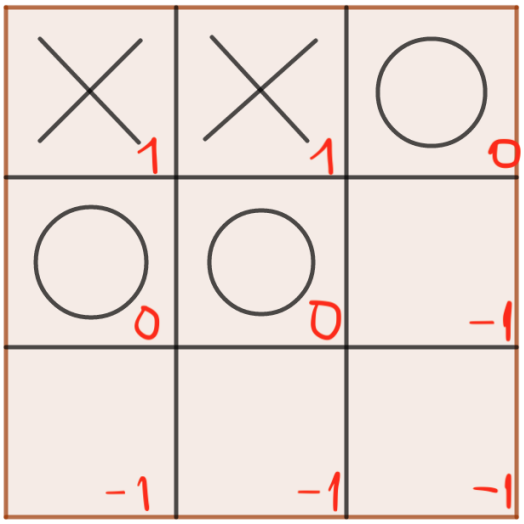
Tableaux à deux dimensions

Le **morpion** est un jeu où deux joueurs peuvent disposer l'un après l'autre un symbole X ou O sur un plateau 3×3, jusqu'à ce que l'un des joueurs a un alignement, horizontal vertical ou diagonal, dans ce cas il a gagné, ou alors le plateau est rempli sans alignement dans ce cas il y a match nul

Il faut distinguer l'aspect graphique constitué du plateau et des symboles de l'aspect logique c'est à dire ce qui se passe dans la mémoire de l'ordinateur au fur et à mesure que le jeu se déroule

Nous avons besoin de maîtriser **les tableaux à 2 dimensions** pour programmer efficacement ce jeu



1 Tableau vs Liste

Jusqu'ici nous avons utilisé des tableaux en Python mais il se trouve que la structure utilisée en Python, appelée liste, est plus "riche" que celle d'un tableau

```
>>> t = [1,2,3]

>>> type(t)
<class 'list'>
```

En effet **dans les langages comme C ou Java un tableau a une taille fixe** alors qu'**en Python on peut faire évoluer la taille d'un tableau (on devrait plutôt dire une liste)**, par exemple en ajoutant un nouvel élément avec la fonction `append()`

```
>>> t.append(4)

>>> t
[1, 2, 3, 4]
```

Ceci nous permet de construire une liste de plusieurs manières :

Si on veut construire une liste contenant les entiers de 1 à 100 on peut faire :

```
liste = []
for i in range(1,101):
    liste.append(i)
```

Ou on peut **construire la liste par compréhension** ainsi

```
liste = [i  for i in range(1,101)]
```

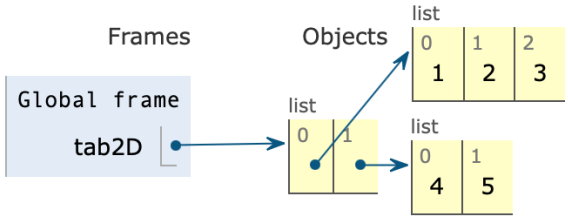
2 Tableau à deux dimensions

Un tableau à deux dimensions est un tableau où chaque cellule est une référence vers un tableau

Par exemple

```
tab2D = [[1, 2, 3], [4, 5]]
```

est visualisé par



Plusieurs questions se posent alors

1. A quoi peuvent servir un tableau à deux dimensions ?
2. Comment **créer** le tableau à l'état initial ?
3. Comment **modifier une cellule** en particulier ?
4. Comment **parcourir un tableau à deux dimensions** ?

2.1 Utilité d'un tableau à deux dimensions

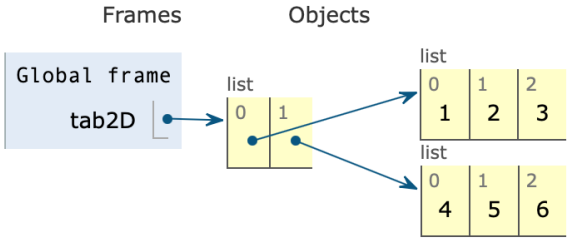
Nous utiliserons cette année à plusieurs reprises des tableaux à deux dimensions par exemple :

1. pour programmer un jeu de plateau
2. pour le traitement de données en tables

Ainsi on considère que

```
tab2D = [[1, 2, 3], [4, 5, 6]]
```

visualisé par



représente les lignes du tableau
1 2 3
4 5 6
plutôt que les colonnes du tableau
1 4
2 5
3 6
qui serait plutôt représenté par

```
tab2D = [[1, 4], [2, 5], [3, 6]]
```

Dans la suite on utilisera la plupart du temps des tableaux dont les lignes sont des tableaux ayant toujours le même nombre d'éléments

Si il y a n lignes et si chaque ligne contient m éléments on parle de tableau $n \times m$

2.2 Création d'un tableau à deux dimensions

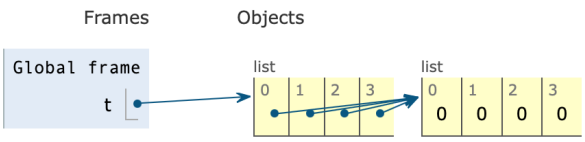
Lorsqu'il y a peu d'éléments on crée la tableau **en extension** (comme précédemment), par contre lorsqu'il y a beaucoup d'éléments, suivant une logique, on va procéder **en compréhension**

Par exemple :

Si on veut créer un tableau de 4 x 4 ne contenant que des 0 on ne va pas procéder ainsi

```
t = [[0]*4]*4
```

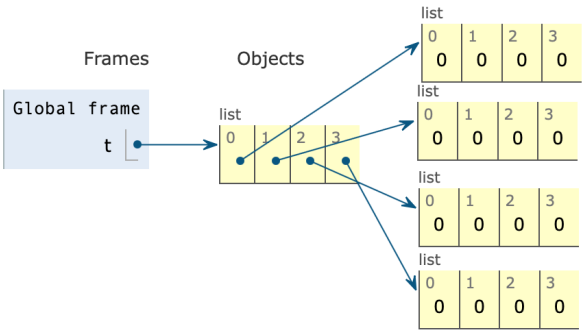
car dans ce cas on aurait ceci



mais comme ceci

```
t = [[0]*4 for i in range(4)]
```

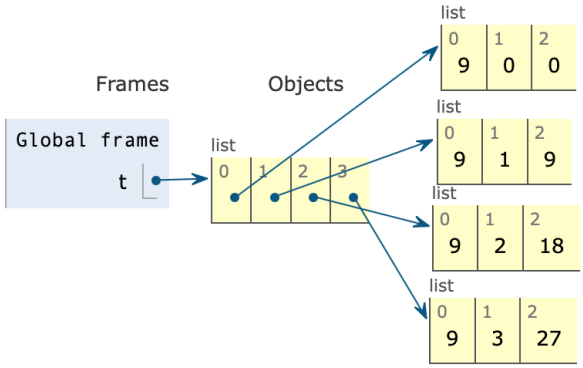
ce qui nous donne



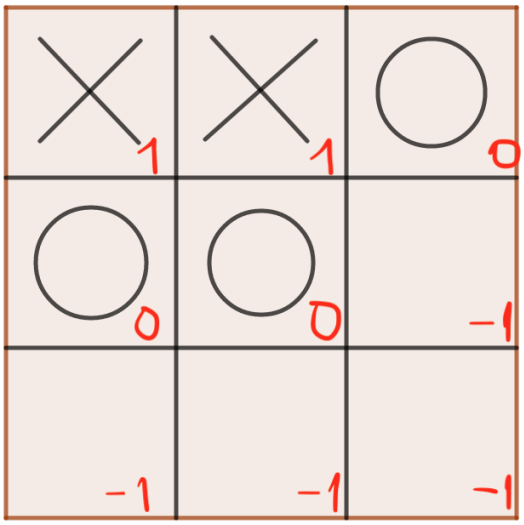
Si on veut créer la table de multiplication de 9

```
t = [[9, i, 9*i]  for i in range(11)]
```

ce qui donne pour les quatre premières lignes



2.3 Modification d'une cellule



Pour le jeu du morpion on va créer un tableau à l'état initial ne contenant que des -1 (représentant une case vide)

```
t = [[-1]*3  for i in range(3)]
```

Supposons que le premier joueur place un symbole O dans la case centrale, ceci se traduit par l'affectation (O est représenté par 0)

```
t [1][1] = 0
```

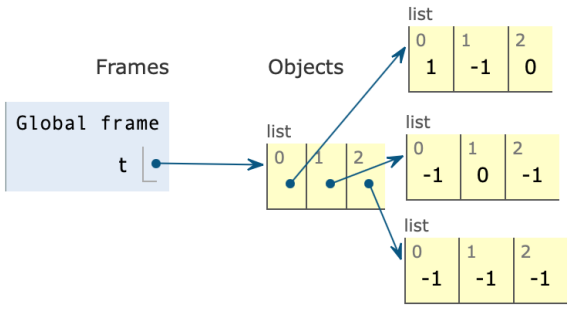
Ensuite le deuxième joueur place un X dans la première case de la première ligne (X est représenté par 1)

```
t [0][0] = 1
```

Ensuite le premier joueur place un O dans la troisième case case de la première ligne

```
t [0][2] = 0
```

Ces trois actions sont visualisées par



Retenir :
Dans le **premier** crochet on met l'indice de la **ligne**
Dans le **deuxième** crochet on met l'indice de la **colonne**

2.4 Parcourir un tableau à deux dimensions

On rappelle qu'il existe deux façons de parcourir un tableau `t` à une dimension

```
for i in range(len(t)):
    #traitement sur t[i]
```

ou

```
for element in t:
    #traitement sur element
```

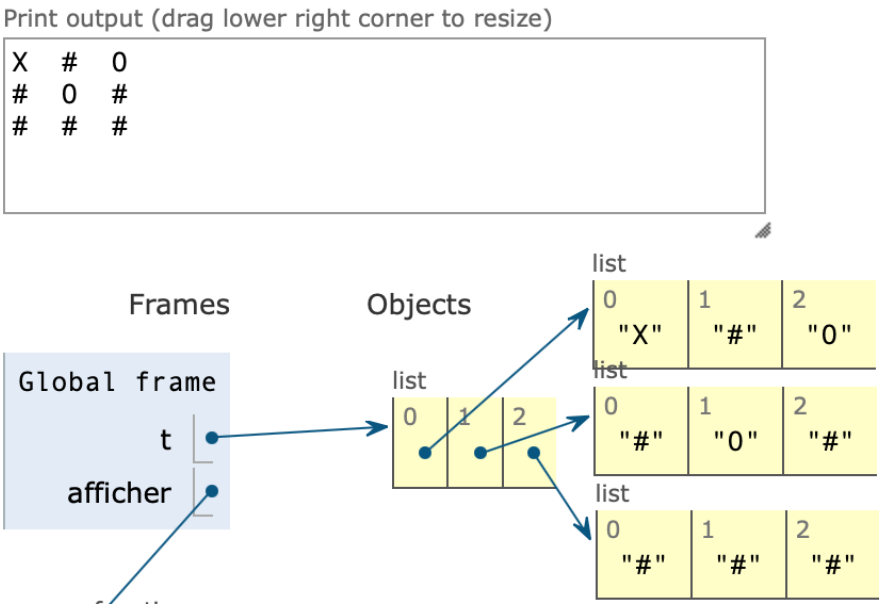
On veut afficher le contenu du tableau à chaque tour du jeu, cette fois ci on met dans une cellule du tableau le caractère "#" si la case en rapport est vide, "X" si on met un X dans la case en rapport ou "O" si on met O dans la case

On va pour cela définir deux fonctions, l'une `affiche_tableau(t)` qui ne retourne rien et qui utilise la fonction `print()` à l'intérieur du corps de la fonction et l'autre `list_to_string(t)` retournant une chaîne de caractères sur laquelle on applique la fonction `print()`

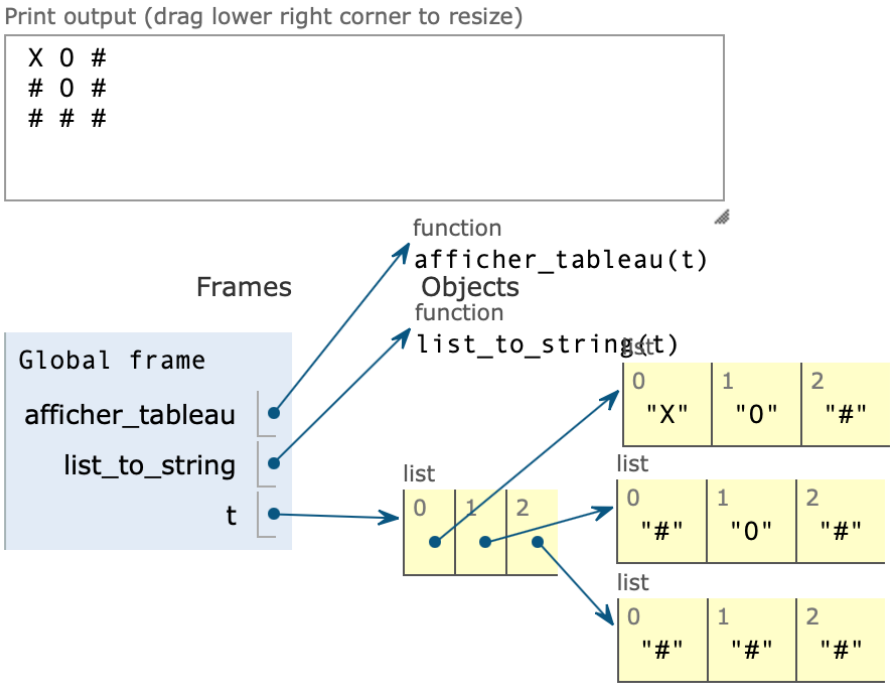
```
def afficher_tableau(t):
    for ligne in t:
        for cellule in ligne:
            print(cellule," ",end="")
```

```
        print()

def list_to_string(t):
    ch = ""
    for ligne in t:
        for cellule in ligne:
            ch = ch + " " + cellule
        ch = ch + "\n"
    return ch

#initialisation
t = [["#"]*3 for i in range(3)]
#on joue 3 coups
t[1][1] = "O"
t[0][0] = "X"
t[0][1] = "O"
#affichage du tableau
#afficher_tableau(t)
print(list_to_string(t))
```

La visualisation donne pour la première fonction



et pour la deuxième fonction



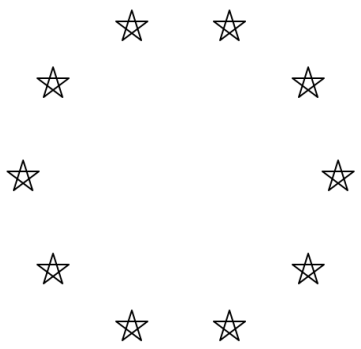
Exercices

Ex 1

Construire **par compréhension**

1. La liste des entiers pairs de 0 jusqu'à 100 compris
2. La liste des carrés des entiers i avec i variant de 1 à 100 compris
3. La liste des cubes des entiers i avec i variant de 1 à 100 compris

Ex 2



Il s'agit de faire dessiner par la Tortue dix étoiles disposées en cercle

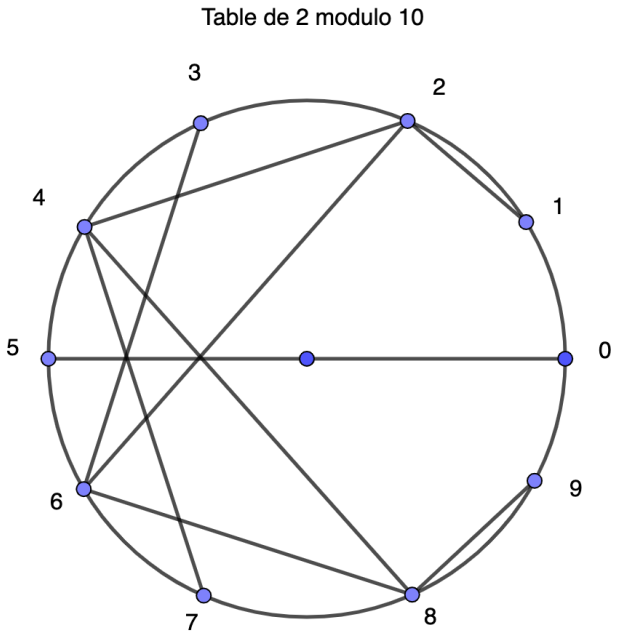
1. Créer une liste angles en compréhension de $\frac{2k\pi}{10}$ pour k variant de 0 jusqu'à 9
2. Créer une liste de **tuples** en compréhension $(100\cos(a), 100\sin(a))$ pour a variant dans la liste angles
3. Finir en itérant sur la liste de tuples

Ex 3

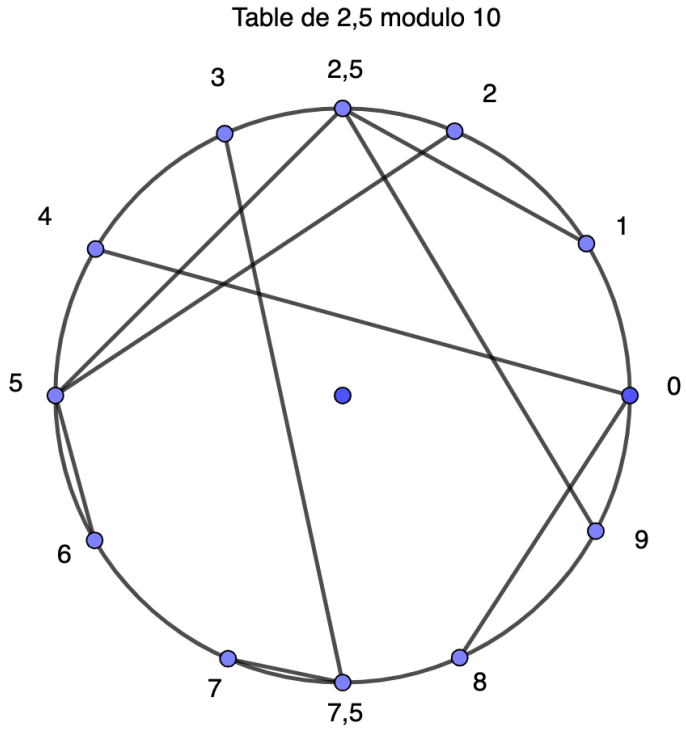
Il s'agit de faire tracer par la tortue les tables de multiplication de n modulo m où n et m sont des entiers dans un premier temps avec n $<$ m (dans un seconds temps n est un nombre décimal)

Regarder ici le principe `https://www.youtube.com/watch?v=-X49VQgi86E`

Par exemple voici le tracé de la table de 2 modulo 10 :



Par exemple voici le tracé de la table de 2,5 modulo 10 :



1. Réutiliser l'exercice 2 pour définir une fonction `table(m : int,n : int) -> None` qui trace avec la tortue la table de m modulo n avec m et n entiers avec m $<$ n (à vous de définir des fonctions auxiliaires)
2. Définir une fonction `table_2(m : float,n : int) -> None` qui trace avec la tortue la table de m modulo n avec m décimal et n entier avec m $<$ n

Ex 4

Voici un programme en Python :

```
regles = [[False]*4]*8
```

```
regles[0][2] = True
print(regles[1][2])
```

Qu'observe-t-on à l'écran ?
A) `True` B) `False`

Ex 5

Voici un programme en Python :

```
regles = [[False]*4 for i in range(8)]
regles[0][2] = True
print(regles[1][2])
```

Qu'observe-t-on à l'écran ?
A) `True` B) `False`

Ex 6

Ecrire une fonction `table(nombre)` où nombre est un entier de 2 à 9 compris et qui renvoie sous forme de tableau 2D 11x 3 la table de multiplication `t` de `nombre` où `t[i][0] = nombre` pour tout i, `t[i][1] = i` pour tout i et `t[i][2] = nombre x i` pour tout i .

Ex 7

Ecrire une fonction `maximum(t)` qui renvoie la valeur maximale contenue dans un tableau à deux dimensions `t` contenant des nombres.

Ex 8

Dans un tableau à deux dimensions chaque ligne est une liste contenant les lettres "A", "T", "C" et "G"

1. Définir une fonction `taux_CG(liste)` qui retourne le taux des lettres C et G pour une liste donnée

   Par exemple

   ```
   liste = ["C", "T", "A", "T", "T","G", "A", "A"]
   >>> taux_CG(liste)
   0.25
   ```

2. Définir une fonction `indice_taux_max(tableau)` qui retourne l'indice de la ligne du tableau ayant le taux de CG maximal (s'il y a plusieurs candidats possibles l'indice retourné est le premier indice)

Ex 9

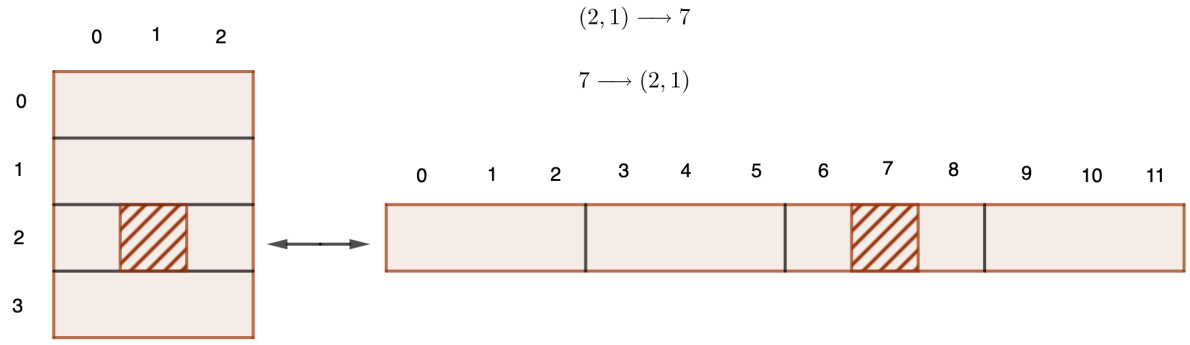
Définir une fonction `tuple_maximum(t)` qui renvoie le tuple (i,j) de la première occurence de la valeur maximale contenue dans un tableau à deux dimensions `t` qui contient des nombres, où i est l'indice de ligne et j l'indice de colonnes

Par exemple

```
liste = [[-2, -3, -4, -1, -2], [0,-3,-1,0,-4]]
>>> tuple_maximum(liste)
(1,0)
```

Ex 10

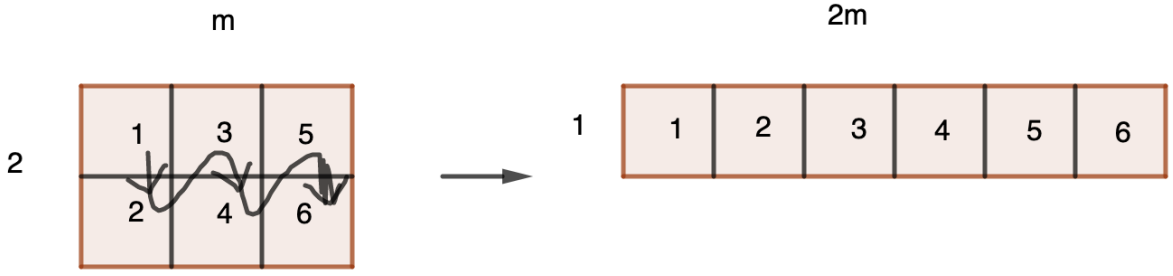
Le but de l'exercice est de convertir un tableau à deux dimensions `tab2D` de taille NB_LIGNES × NB_COLONNES en un tableau à une dimension `tab1D` de taille NB_LIGNES * NB_COLONNES et vice-versa



1. Comprendre la logique du procédé à partir du dessin ci-dessus et définir `tab2D_1D(tab2D)` qui renvoie un tableau 1D obtenu à partir du tableau 2D `tab2D` à partir du procédé
2. Ecrire `tab1D_2D(tab1D)` qui renvoie un tableau 2D obtenu à partir du tableau 1D `tab1D` à partir du procédé

Ex 11

Etant donné deux tableaux t1 et t2 de même taille écrire une fonction `fusionner(t1,t2)` qui renvoie un tableau de taille double que celle de t1 et de telle sorte que les éléments de t1 et t2 sont répartis dans le nouveau tableau selon le schéma suivant



Ex 12

Etant donné un tableau 2D `tab` n x m ayant un nombre n pair de lignes, écrire une fonction Python `etirer(tab)` qui renvoie un tableau 2D $\frac{n}{2}$ x 2m construit ainsi :

On groupe les lignes du tableau `tab` deux par deux et on les fusionne selon l'exercice 11

```
tab =[[1, 3, 5],
      [2, 4, 6],
      [7, 9, 11],
      [8, 10, 12]]

n_tab = [[1, 2, 3, 4, 5, 6],
         [7, 8, 9, 10, 11, 12]]
```

Ex 13 Ecrire une fonction `renverser(tab)` qui prend en paramètre un tableau `tab` et qui renvoie un nouveau tableau ayant les mêmes éléments que `tab` mais dans l'ordre inverse

```
>>> tab = [1,2,3]
>>> renverser(tab)
[3, 2, 1]
```

Ex 14

Etant donné un tableau 2D `tab` n x m ayant un nombre pair de colonnes écrire une fonction Python `replier(tab)` qui renvoie un tableau 2D 2n x $\frac{m}{2}$ construit ainsi :

On coupe le tableau dans le sens de la hauteur par le milieu, la partie droite est repliée sous la partie gauche (on renverse les lignes et les colonnes)

```
tab = [[1, 2, 3, 4, 5, 6],
       [7, 8, 9, 10, 11, 12]]


n_tab = [[1, 2, 3],
         [7, 8, 9],
         [12, 11, 10],
         [6, 5, 4]]
```

Ex 15

La transformation du boulanger sur un tableau 2D n x n ayant un nombre pair de lignes consiste à réaliser les opérations étirer et replier successivement

1. Ecrire une fonction `boulanger(tab)`
2. En partant d'un tableau 2D carré de taille 2 x 2 on réitère la transformation du boulanger. Au bout de combien de fois obtient le tableau de départ (le faire à la main)

Ex 16 Météo France a installé des balises en mer relevant par exemple la hauteur des vagues toutes les 20 minutes

1. Définir par compréhension la liste `minutes` des minutes écoulées à partir de minuit par tranche de 20 minutes jusqu'à minuit prochain inclus

   Le début du tableau est `[0, 20, 40, 60, 80, ...`

2. Définir une fonction `temps(minutes)` qui retourne une chaîne de caractères `minutes` en un temps exprimé en heure et minutes

   Par exemple

   ```
   >>> temps(80)
   '1:20'
   ```

3. Définir par compréhension un tableau à deux dimensions `hauteur_vagues`, où chaque ligne est constituée d'abord de l'heure sous la forme d'une chaîne de caractères comme précédemment, puis de la hauteur de vague

   La hauteur de la vague est de type `float` avec un chiffre après la virgule.

On engendrera les hauteurs de manière aléatoire avec la fonction `random.uniform(1,7)` qui retourne un nombre à virgules compris entre 1 et 7. Pour arrondir on utilisera la fonction `round(...)` (Faire `help(round)`)

4. Définir une fonction `filtre(tableau,seuil)` qui à partir d'un tableau de mesures retourne un tableau à deux dimensions constitué des heures où les hauteurs de vagues sont supérieures ou égales à la valeur du `seuil`