

# Tp - Redimensionnement d'une photo – Suite

Ce Tp prolonge le tp1 sur le redimensionnement intelligent. Il propose de reprendre le code mis au point dans le tp1 pour l'insérer dans une interface Tkinter en P.O.O.

## 1- DEMARRAGE DU TP :

⇒ Télécharger sur [nsibranly.fr](http://nsibranly.fr), le dossier *redim.zip* . Extraire les 3 fichiers qu'il contient dans votre zone de travail :

- *bibliRedim.py* contient le code de la classe de la classe *Redim()*. Cette classe permet entre autres de générer aléatoirement la liste de listes « *pixel* » et ensuite de la visualiser sur une interface Tkinter, le chemin de poids minimum. Ce fichier n'est pas à modifier.



- *redimensionnement.py* contient le code de la classe *RedimTp()* qui hérite des méthodes de la classe *Redim()*. C'est uniquement dans ce fichier que vous ajouterez du code.

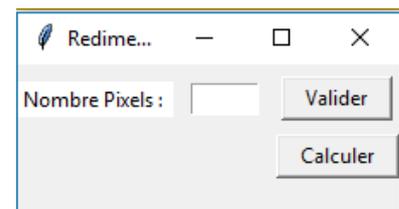
- *image.jpg* qui est une image de 30px par 30px.

L'objectif de ce tp2 est de rajouter des méthodes dans la classe *RedimTp()* afin de pouvoir déterminer le chemin de poids minimum.

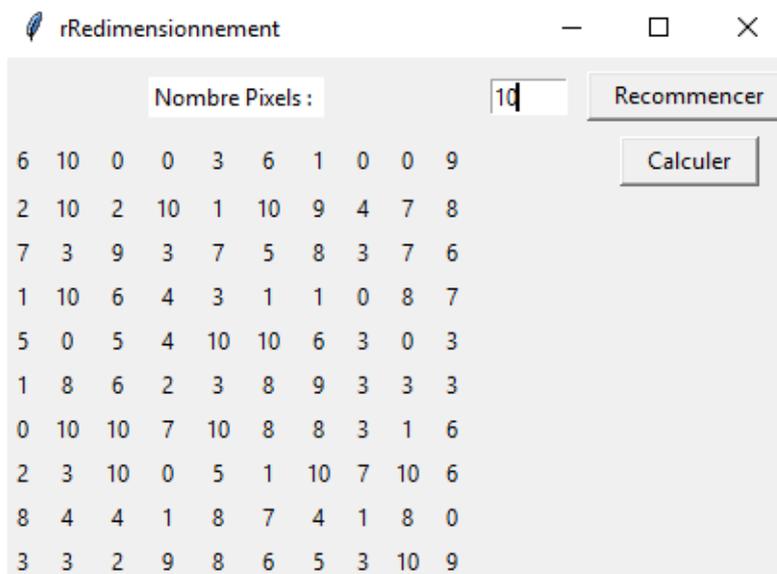
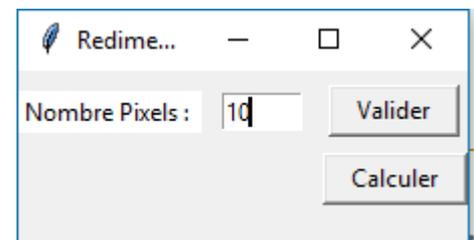
⇒ Ouvrir le fichier *redimensionnement.py* .

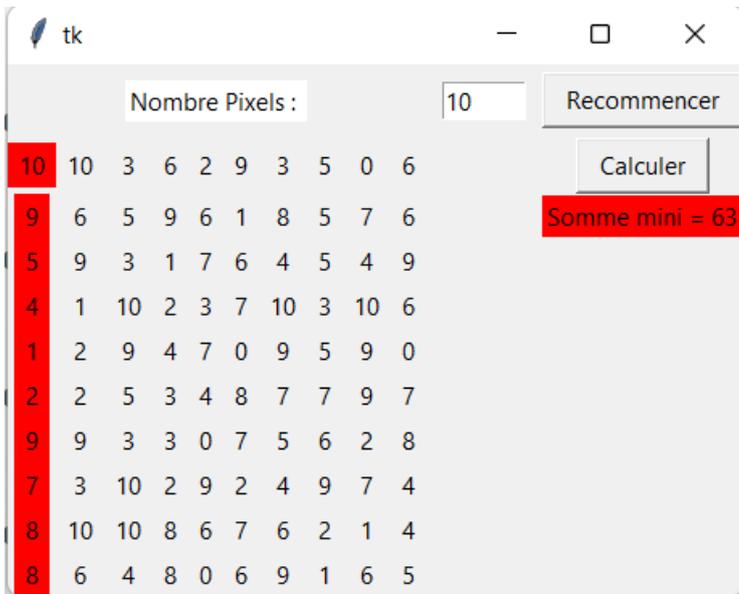
⇒ Exécuter le code

..... la fenêtre Tkinter suivante apparaît :



En saisissant une valeur de 10 et en cliquant sur *Valider*, un tableau de 10 lignes et 10 colonnes rempli de nombres entiers aléatoires compris entre 0 et 10 apparaît :





En cliquant sur *Recommencer* , en ayant modifié ou pas le nombre de pixels, un nouveau tableau est généré.

En cliquant sur *Calculer* , la colonne de gauche devient rouge pour l'instant.

## 2- PRISE EN MAIN DU CODE FOURNI :

Les différentes parties du code déjà écrit sont détaillées ci-dessous :

```
from bibliRedim import Redim
```

La classe RedimTp hérite des méthodes de la classe Redim

```
class RedimTp(Redim) :
```

```
    def __init__(self):
        Redim.__init__(self)
```

En créant une instance de la classe RedimTp, le constructeur de la classe Redim est exécuté. Celui-ci crée entre autres, la fenêtre Tkinter de départ.

```
    def getPixel(self) :
        return self.pixel
```

```
    def setResultat(self, liste) :
        self.resultat = liste
```

Cette méthode retourne la liste de listes « **pixel** » qui est générée aléatoirement et affichée sur l'interface Tkinter. Cette liste est contenue dans l'attribut *pixel* de l'objet.

```
    def redimensionnement(self) :
        pix = self.getPixel()
        n = len(pix)
        resultat = [0 for i in range(n)]
```

Cette méthode permet de modifier l'attribut *resultat* de l'objet qui contient la liste « **resultat** ». Cette liste permet de colorier en rouge le chemin de poids minimum sur l'interface Tkinter.

```
        self.setResultat(resultat)
        self.redWay()
```

```
# Main
```

```
R = RedimTp()
```

```
R.mainloop()
```

La méthode `redimensionnement()` est exécutée au clic sur le bouton **Calculer**.

Pour l'instant, elle permet simplement :

- De récupérer la liste de liste « **pixel** » nommée ici `pix[]` et de l'afficher dans la console
- De générer une liste `resultat[]` initialisée à 0
- De modifier l'attribut `resultat` sur l'objet
- De colorer le chemin en rouge sur l'interface Tkinter (méthode `redWay()`)

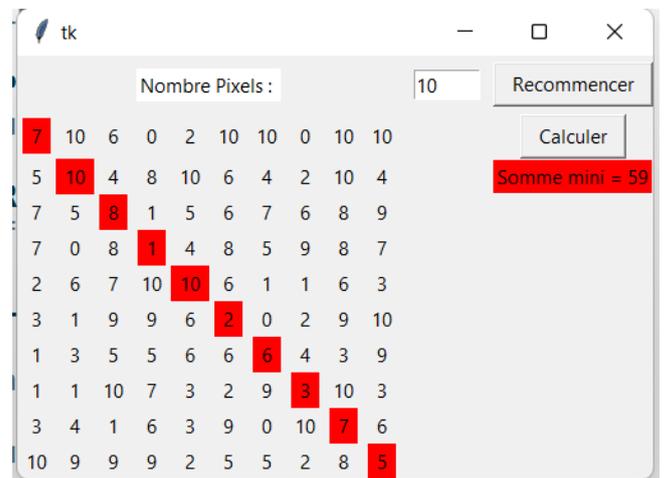
```
def redimensionnement(self) :
    pix = self.getPixel()
    print(pix)
    n = len(pix)
    resultat = [0 for i in range(n)]

    self.setResultat(resultat)
    self.redWay()
```

⇒ Pour prendre en main ce code, modifier par exemple les valeurs d'initialisation de la liste `resultat` :

```
resultat = [i for i in range(n)]
```

En exécutant alors le script, après clic sur le bouton **Calculer**, on obtient :



### 3- CREATION DU CODE :

⇒ Compléter la classe `RedimTP` afin qu'elle puisse afficher le chemin de poids minimum. Il s'agit de compléter la méthode `redimensionnement()` et d'en créer d'autres qui permettront au final, de générer une liste de listes « **memo** » et à partir de celle-ci, de générer la liste « **resultat** ». Il s'agit bien sûr de reprendre les scripts des fonctions `optimalIter()`, `indice_min()`, `solution()` du tp1 et qui ont été écrits en programmation fonctionnelle. Il sera nécessaire de les adapter à la POO et au nouveau contexte de ce tp. Pour débayer, travailler sur une image de 3 pixels, pas plus.

### 4- TEST DU CODE ECRIT :

⇒ Tester le code pour différentes valeurs de `n` : `n = 2`, `n = 8`, `n = 20`, `n = 40`. La valeur maximale de `n` a été bridée à 40.

## 5- TEST DU CODE SUR LE CAS D'UNE IMAGE :

On se propose ici d'utiliser le code réalisé sur une liste de listes *pixel* composés de poids obtenus à partir de l'image ci-contre, de dimension carrée.



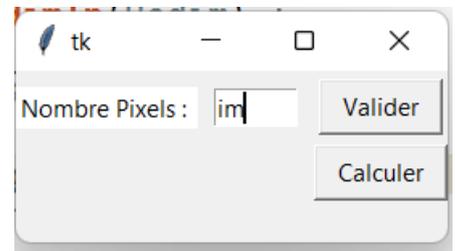
L'interface Tkinter ne supportant pas un traitement pour un nombre *n* important de pixels, on travaillera ici sur la seconde image donnée à gauche et de taille 30 x 30 pixels uniquement.



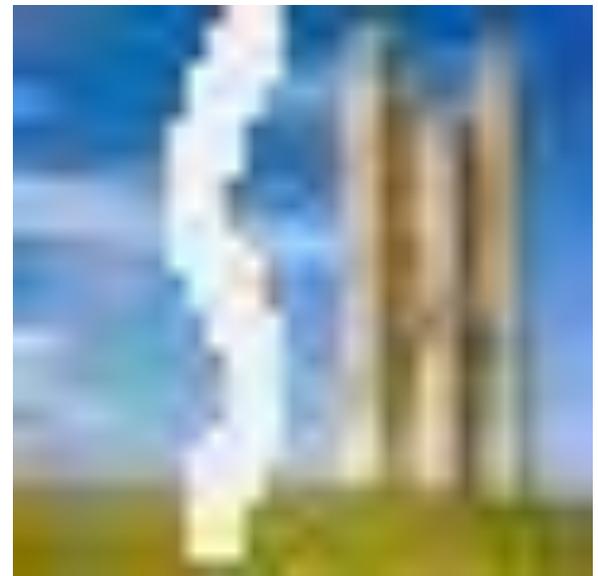
Les poids utilisés correspondront au complémentaire à 256 de la moyenne des valeurs R V B de chaque pixel. Ces poids correspondent ainsi au complémentaire de l'intensité en niveau de gris de chaque pixel. De plus un trait clair a été tracé à main levée à travers l'image afin d'obtenir des poids plus faibles dans le tableau.

⇒ Ouvrir le fichier *image.jpg* avec un visionneur d'image pour vérifier qu'il contient bien l'image ci-dessus.

⇒ Exécuter votre code et en saisissant cette fois ci, non plus le nombre *n* de pixels de la liste « *pixel* », mais le mot clé *im*. En indiquant ce mot clé, la liste « *pixel* » est créée par lecture des valeurs *rgb* de chacun des 30x30 pixels contenus dans le fichier *image.jpg*.



⇒ Cliquer sur *Valider*, puis sur *Calculer* ...



... on retrouve ainsi la trace du trait réalisé dans l'image.

## 6- CONCLUSION :

Ce TP a permis d'implémenter l'algorithme de recherche du chemin de poids minimum, en utilisant un paradigme de programmation dynamique. Il a été utilisé pour trouver un chemin sur une image au format *.jpg*.

Le paradigme de programmation dynamique utilisé permet de réduire fortement le volume de calculs et donc la complexité du problème, tout en permettant d'obtenir une solution exacte.

**Pour une image carrée de 30px , une recherche optimale, en prenant en compte tous les cas de figures, conduirait à une complexité exponentielle  $\mathcal{O}(3^n)$ . Le calcul de  $3^{30}$  donne environ 200 000 milliards de cas à étudier. En adoptant une programmation dynamique, la complexité devient linéaire et les temps de calculs deviennent instantanés.**

Il serait intéressant d'utiliser ce principe pour l'appliquer sur les problèmes de redimensionnement intelligent d'images. Mais cela sort du cadre de ce Tp.