**Mise en place d’un outil de suivi d’imagerie clinique**

**Notice du code**

**Auteur : Kenza Benkirane**

**Tuteur : Stan Durand**

**Table des matières**

[I. Plan du code 3](#_Toc49007472)

[1. Importations et modules 3](#_Toc49007473)

[2. Variables et options 3](#_Toc49007474)

[3. Directories à modifier 4](#_Toc49007475)

[II. Interface graphique 4](#_Toc49007476)

[1. Pourquoi Tkinter ? 4](#_Toc49007477)

[2. Créer une page : le constructeur 4](#_Toc49007478)

[III. Fonctions 8](#_Toc49007479)

[1. Openradio : Page de saisi d’informations sur image 8](#_Toc49007480)

[2. Fonctions d’insertion d’informations sur image 13](#_Toc49007481)

[3. Fonction database 21](#_Toc49007482)

[IV. Propositions d’amélioration et idées 22](#_Toc49007483)

[ Fonctions zoom in et out 23](#_Toc49007484)

**Table des figures**

[Figure 1 : Directories pour le logo 4](#_Toc49007485)

[Figure 2 : Page d'accueil de l'application 7](#_Toc49007486)

[Figure 3 : Architecture des boutons et fonctions associées 8](#_Toc49007487)

[Figure 4 : Deuxième Page de l'application 9](#_Toc49007488)

[Figure 5 : Boite d'ouverte de fichier générée par askopenfilename 10](#_Toc49007489)

[Figure 6 : Affichage dans la frame 2 f2 en Page de traitement d’images 12](#_Toc49007490)

# Plan du code

## Importations et modules

﻿from tkinter import \* ; import tkinter as tk

from PIL import ImageTk as itk; from PIL import Image

import subprocess; import os ;

import cv2

import numpy as np

import pandas as pd;from pandastable import Table,TableModel;

* Le module Tkinter sert pour *l’Interface graphique.* Utiliser le diminutif « tk » permet d’éviter les erreurs mineures et de reconnaître les modules Tkinter. Une aide python est disponible [ici.](https://docs.python.org/fr/3/library/tkinter.html)
* Le module PIL (pillow) permet d’importer des images et de les inclure dans des canvas Tkinter, d’où le itk : ImageTk
* Les modules subprocess et os permettent de rechercher des fichiers dans un dossier
* CV2 correspond à Open CV qui sert pour le traitement d’image. Il n’a pas été utilisé dans la suite du code mais peut s’avérer utile pour une modification postérieure.
* Le module numpy est un module fondamental de python qui est souvent nécessaire pour exécuter d’autres modules. Il sert principalement pour la manipulation de matrices ou tableaux.
* Les modules pandas et pandastable servent à la création de bases de données à partir du traitement d’image.

## Variables et options

﻿TITLE\_FONT = ("Arial", 25, "bold")

violet\_bg='#ccb6e4'

white\_bg='#FFFFFF'

cross\_color=(255,255,255)

V=['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7','T1','T2','T3','T4','T5','T6','T7','T8','T9''T10','T11','T12','L1','L2','L3','L4','L5']

Titre\_Application="X-Rays Measurements Tracking Platform"

from numpy import \*

global radio

radio=None

Ces variables sont utilisées à plusieurs reprises pour l’esthétique de l’application et

## Directories à modifier

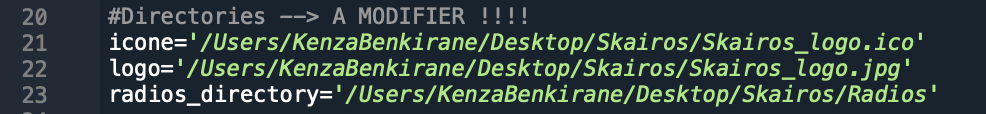


Figure 1 : Directories pour le logo

Ces directories permettent de donner un logo à l’application, mais aussi de mettre le logo de Skairos sur l’ensemble des pages de l’application. Lors de la transformation de l’application en fichier exécutable, ce logo sera intégré directement dans le dossier ‘read me’ et n’aura pas besoin d’être stocké à l’extérieur

# Interface graphique

## Pourquoi Tkinter ?

Le plugin choisit pour l’interface graphique est Tkinter, pour les raisons suivantes :

* Simplicité d’utilisation : il a déjà été utilisé et pourra être réutilisé assez rapidement afin de pouvoir gagner du temps sur les étapes supérieures
* Disponible en open source
* Très souvent utilisé donc il existe énormément de ressources et références sur internet qui peuvent être utiles en cas de doute

## Créer une page : le constructeur

﻿

class Application(object):

"""Défini l'objet application qui est notre application de suivi d'imageries cliniques"""

def \_\_init\_\_ (self):

"""Construteur de la fenêtre principale"""

#Paramètres de la fenêtre

self.fen=Tk()

global frame; frame=self.fen

self.fen.configure(background='white')

self.fen.title(Titre\_Application)

self.fen.iconbitmap(icone)

self.fen.minsize(1000,400)

﻿#Affichage du logo sur toutes les pages

global image;global photo

image = Image.open(logo)

image = image.resize((80, 50), Image.ANTIALIAS)

photo = itk.PhotoImage(image)

label = Label(frame,image=photo,bg=white\_bg)

label.image = photo # keep a reference!

label.pack(side='bottom', fill='x')

self.fen.iconphoto(False,photo)

self.fen.iconbitmap(icone)

La classe Application permet de créer l’objet Application. A celui ci est toujours associé le constucteur, la fonctio\_\_init\_\_, dans laquelle on définit les éléments de base de notre objet :

* Il s’agit d’un GUI Tkinter : self.fen =Tk()
* frame=self.fen : ce nom est utilisé par soucis de simplification
* On utilise le titre de l’application défini précédemment
* On lui donne un icône
* On donne une taille minimale à la fenêtre : self.fen.minsine(longueur,largeur)
* Donner un icône à l’application [self.fen.iconephoto(False,file=’yourfile.png’)](https://www.geeksforgeeks.org/iconphoto-method-in-tkinter-python/), en utilisant la fonction [PhotoImage](https://effbot.org/tkinterbook/photoimage.htm)

On pourrait aussi :

* Lui donner une taille maximale : ﻿self.fen.maxsize(1000,500)

#Initialisation des frames

global f; f = tk.Frame(self.fen,bg=white\_bg) #frame pour la base de données

global f2;f2 = tk.Frame(self.fen,bg=white\_bg) #frame pour les boutons

f.pack(fill='y',expand=0,side='right')

f2.pack(fill='y',expand=0,side='left')

Les deux frames f et f2 serviront à « porter » les boutons de contrôle ainsi que les tableaux de coordonnées. Ici expand = 0 pour qu’elles ne s’affichent que si un bouton y est « packed », ce qui n’a lieu que lorsque l’on choisit une radio, soit après avoir appuyé sur le bouton « Parcourir », donc après avoir activé la fonction « openradio ».

La fonction global permet de les utiliser hors de la fonction \_\_init\_\_, c’est à dire de faire de ces frames des variables globales.

tk.Label(frame, text="\n Plateforme de suivi d'imageries cliniques", font=("Arial", 20, "bold"),bg=white\_bg).pack(fill='x')

﻿global Description\_lbl; Description\_lbl=tk.Label(frame, text="Description \n", font=("Arial", 12, "italic"),bg=white\_bg); Description\_lbl.pack(expand='yes')

self.drawing\_tool = "line" #initalisation du drawing\_tool : paramètre par défaut : ligne

global Start\_bttn; Start\_bttn= tk.Button(frame,text="Commencer",command=lambda:start(self), bg=white\_bg)

Start\_bttn.pack(expand='yes')

Une image contenant capture d’écran, oiseau

Description générée automatiquement

Figure 2 : Page d'accueil de l'application

Les fonctions Label permettent d’afficher du text, \n permet de sauter du ligne.

Nommer les label permet de les supprimer ensuite : Le label « Description\_lbl » sera supprimé lorsque l’on appuie sur le bouton  Parcourir , à l’aide de la méthode Description\_lbl.pack\_forget()

Ici on initialise la variable self.drawing\_tool en lui associant la valeur ‘line’ comme paramètre de défaut : dès qu’on ouvrira l’image, on pourra lui déssiner des lignes dessus

*Le bouton start Permet d’acceder à la page de déscription des étapes. Il affichera le bouton parcourir et tous les textes en conséquence*

Figure 3 : Architecture des boutons et fonctions associées

# Fonctions

## Openradio : Page de saisi d’informations sur image

Cette fonction s’exécute lorsque l’on appuie sur le bouton Parcourir sur la deuxième page.

Une image contenant capture d’écran, oiseau

Description générée automatiquement

Figure 4 : Deuxième Page de l'application

﻿global openradio

def openradio(self):

"""S'execute lorsque l'on appuie sur le bouton "Parcourir" : affiche l'image, créer les boutons pour choisir les formes à créer et la base de données à afficher """

#Visuel de la page modifié

Choisir\_Label.pack\_forget(); etapes.pack\_forget(); #On supprime l'explication des étapes

Parcourir['text'] = 'Changer de radiographie' # "Parcourir" devient "Changer de radiographie" mais conserve la même fonction

global Database\_bttn

Database\_bttn=Button(frame,text='Afficher le tableau des données',bg=white\_bg,command=lambda: database(self))

Database\_bttn.pack(); basededonnee=1;

##### Ouvrir la radio et l'afficher

radio\_filename = tk.filedialog.askopenfilename(initialdir=radios\_directory, title="Select A File", filetypes=(("jpg files", "\*.jpg"),("all files", "\*.\*")))

La ligne précédente va générer une fenêtre pour ouvrir un fichier

Une image contenant capture d’écran

Description générée automatiquement

Figure 5 : Boite d'ouverte de fichier générée par askopenfilename

radio\_img=cv2.imread(radio\_filename)

radio\_img\_tk= Image.fromarray(radio\_img) # Convert the Image object into a TkPhoto object

radio = itk.PhotoImage(image=radio\_img\_tk)

Les lignes précédentes permettent d’afficher l’image choisie dans le canva créée, intitulé drawing\_area.

global drawing\_area

drawing\_area = tk.Canvas(frame, width=radio.width(), height=radio.height(),bg='black') #à améliorer

global Image\_to\_canva;

Image\_to\_canva=drawing\_area.create\_image(0, 0,image=radio,anchor="nw") #Associate the image to the canva

drawing\_area.bind("<Motion>", self.motion)

drawing\_area.bind("<ButtonPress-1>", self.left\_but\_down)

drawing\_area.bind("<ButtonRelease-1>", self.left\_but\_up)

Les trois lignes précédentes permettent d’associer :

* Le mouvement de la souris à la fonction self.motion 🡪 <Motion> : [voir lien](https://www.python-course.eu/tkinter_events_binds.php)
* Le fait d’appuyer sur la souris à la fonction self.left\_but\_down
* Le fait de lâcher la souris à la fonction self.left\_but\_up

drawing\_area.pack(side='top',fill="y")

#Ajout des boutons de contrôle pour choisir la forme à dessiner

global DTool; DTool=tk.Label(f2, text=" \n L'outil de dessin est : \n"+str(self.drawing\_tool)+"\n", font=("Arial", 15),bg=white\_bg)

DTool.pack(side="top")

Button(f2, text="Lignes",font=("Arial", 12),bg=white\_bg,command=lambda: setline(self)).pack()

Button(f2, text="Arc",font=("Arial", 12),bg=white\_bg,command=lambda: setarc(self)).pack()

Button(f2, text="Oval",font=("Arial", 12),bg=white\_bg,command=lambda: setoval(self)).pack()

Button(f2, text="Rectangle",font=("Arial", 12),bg=white\_bg,command=lambda: setrectangle(self)).pack()

Button(f2, text="Point",font=("Arial", 12),bg=white\_bg,command=lambda: setpoint(self)).pack()

Label(f2, text="",bg=white\_bg).pack()

Button(f2, text="Crayon",font=("Arial", 12),bg=white\_bg,command=lambda: setpencil(self)).pack()

Button(f2, text="Legender les vertèbres",font=("Arial", 12),bg=white\_bg,command=lambda: settext(self)).pack()

Cette partie sert à afficher les boutons de contrôle des fonctions associés aux differentes formes, et donc d’obtenir l’affichage suivant :

Une image contenant capture d’écran

Description générée automatiquement

Figure 6 : Affichage dans la frame 2 f2 en Page de traitement d’images

#On initalise les base de données

global coordo\_pointeur; coordo\_pointeur=[] #Coordonnées des points

global coordo\_ligne; coordo\_ligne=[] #Coordonnées des lignes

global coordo\_oval; coordo\_oval=[] #Coordonnées des cercles : centre du cercle pour l'instant

global coordo\_rectangle; coordo\_rectangle=[] #Coordonnées des rectangles

global coordo\_arc; coordo\_arc=[] #Coordonnées des arc

Permet d’initaliser les listes. Celles-ci sont le fondement des bases de données qui constitueront les tableaux qui seront affichés dans l’application.

## Fonctions d’insertion d’informations sur image

global setline

def setline(self):

self.drawing\_tool = "line"

global setarc

def setarc(self):

self.drawing\_tool = "arc"

global setoval

def setoval(self):

self.drawing\_tool = "oval"

global setrectangle

def setrectangle(self):

self.drawing\_tool = "rectangle"

global settext

def settext(self):

self.drawing\_tool = "text"

global setpoint

def setpoint(self):

self.drawing\_tool="point"

global setpencil

def setpencil(self):

self.drawing\_tool="pencil"

Ces fonctions sont associées à tous les boutons de la figure : ils permettent de choisir la figure à dessiner. Ils sont donc associée à la variable self.drawing\_tool qui change de valeur, qui associé à la fonction left\_but\_up suivante :

﻿

def left\_but\_up(self, event=None):

#Méthode s'executant lorsque l'on lache la souris, qui permet de prendre les coordonnées du point où on lache la souris.

self.left\_but = "up"

# Reset the line

self.x\_pos = None

self.y\_pos = None

# Set x & y when mouse is released

self.x2\_line\_pt = event.x

self.y2\_line\_pt = event.y

# If mouse is released and line tool is selected

# draw the line

if self.drawing\_tool == "line":

self.line\_draw(event)

elif self.drawing\_tool == "arc":

self.arc\_draw(event)

elif self.drawing\_tool == "oval":

self.oval\_draw(event)

elif self.drawing\_tool == "rectangle":

self.rectangle\_draw(event)

elif self.drawing\_tool == "text":

self.text\_draw(event)

elif self.drawing\_tool=="point":

self.pointeur(event)

Elle prend pour argument event : le cas où où lache la souris, et activera une des fonctions suivantes :

﻿# ---------- DRAW PENCIL ----------

def pencil\_draw(self, event=None):

if self.left\_but == "down":

# Make sure x and y have a value

if self.x\_pos is not None and self.y\_pos is not None:

event.widget.create\_line(self.x\_pos, self.y\_pos, event.x, event.y, smooth=TRUE,fill="white")

self.x\_pos = event.x

self.y\_pos = event.y

# ---------- DRAW LINE ----------

def line\_draw(self, event=None):

# Shortcut way to check if none of these values contain None

if None not in (self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt):

event.widget.create\_line(self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt, smooth=TRUE, fill="red")

#Database Lignes

coordo\_ligne.append([self.x1\_line\_pt,self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt])

self.df\_lines=pd.DataFrame(coordo\_ligne,columns=['x1','y1','x2','y2'])

self.df=self.df\_lines

# ---------- DRAW ARC ----------

def arc\_draw(self, event=None):

# Shortcut way to check if none of these values contain None

if None not in (self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt,self.y2\_line\_pt):

coords = self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt,self.y2\_line\_pt

# start : starting angle for the slice in degrees

# extent : width of the slice in degrees

# fill : fill color if needed

# style : can be ARC, PIESLICE, or CHORD

event.widget.create\_arc(coords, start=0, extent=150,outline='white',

style=ARC)

#Database Arc

coordo\_arc.append([self.x1\_line\_pt, self.y1\_line\_pt,self.x2\_line\_pt, self.y2\_line\_pt])

self.df\_arc=pd.DataFrame(coordo\_arc,columns=['x1','y1','x2','y2'])

self.df=self.df\_arc

# ---------- DRAW OVAL ----------

def oval\_draw(self, event=None):

if None not in (self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt):

# fill : Color option names are here http://wiki.tcl.tk/37701

# outline : border color

# width : width of border in pixels

event.widget.create\_oval(self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt,

fill="blue",

outline="white",

width=2)

#Database Cercle

coordo\_oval.append([self.x1\_line\_pt, self.y1\_line\_pt])

self.df\_oval=pd.DataFrame(coordo\_oval,columns=['x','y'])

self.df=self.df\_oval

# ---------- DRAW RECTANGLE ----------

def rectangle\_draw(self, event=None):

if None not in (self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt,self.y2\_line\_pt):

event.widget.create\_rectangle(self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt,

outline="white",

width=2)

#Database Rectangle

coordo\_rectangle.append([self.x1\_line\_pt, self.y1\_line\_pt,self.x2\_line\_pt, self.y2\_line\_pt])

self.df\_rectangle=pd.DataFrame(coordo\_rectangle,columns=['x1','y1','x2','y2'])

self.df=self.df\_rectangle

# ---------- DRAW TEXT ----------

def text\_draw(self, event=None):

k=0

if None not in (self.x1\_line\_pt, self.y1\_line\_pt):

event.widget.create\_text(self.x1\_line\_pt, self.y1\_line\_pt,

fill="white",

font=("Arial", 10),

text=V[self.k])

self.k+=1

# ---------- DRAW POINTS ----------

def pointeur(self,event=None):

event.widget.create\_oval(self.x1\_line\_pt, self.y1\_line\_pt, self.x2\_line\_pt, self.y2\_line\_pt,

fill="white",

outline="white",

width=1)

#Database Pointeur

coordo\_pointeur.append([self.x1\_line\_pt, self.y1\_line\_pt]) #ajoute les coordonnées des points à la liste

self.df\_points=pd.DataFrame(coordo\_pointeur,columns=['x','y'])

self.df=self.df\_points

A la fin de chaque base de données, deux lignes permettent d’ajouter les nouveaux widget déssinés à la liste correspondante, puis à la dataframe qui sera affichée grace à la fonction database.

Les paramètres des formes sont modifiables :

* Pour la couleur, il suffit de choisir une autre couleur à la place de white : toutes les couleurs possibles sont disponibles [ici](https://wiki.tcl-lang.org/page/Color+Names%2C+running%2C+all+screens) : ﻿<http://wiki.tcl.tk/37701>
* outline représente la couleur du bord
* On peut aussi choisir la couleur avec les HTML colors RGB, en mettant (255,255,255) par exemple pour le blanc. Vous pouvez trouver la votre [ici](https://htmlcolorcodes.com/): <https://htmlcolorcodes.com>
* ﻿Pour la largeur, width represente la largeur des bords, en pixels
* Pour les arc,
  + le style peut : ARC, PIESLICE, or CHORD. A vous de tester les différentes possibilités pour choisir celle qui vous convient.
  + ﻿extent : width of the slice in degrees
  + ﻿ start : starting angle for the slice in degrees

## Fonction database

﻿ global database

def database(self):

Database\_bttn['text'] = 'Mettre les données à jour'

DTool['text']="\n Les données affichées \n dans le tableau sont : \n "+self.drawing\_tool+"\n"

print(self.drawing\_tool)

print(self.df)

# tk.Label(f, text="Table of the"+self.drawing\_tool, font=("Arial", 11),bg=white\_bg).pack()

self.table = pt = Table(f, dataframe=self.df,

showtoolbar=True, showstatusbar=True)

pt.show()

La base de données affichées est celle pour laquelle on a déssiné un objet pour la dernière fois

# Propositions d’amélioration et idées

* La Scrollbar

Elle permetterait de voir l’ensemble de la radio et de

Voilà une source de code qui peut être utile pour créer une scrollbar :

﻿ # canvas=tk.Canvas(SaisiInfo\_frame,bg=white\_bg)

# canvas.config(scrollregion=canvas.bbox('all'))

# yscrollbar = tk.Scrollbar(self,orient='vertical')

# xscrollbar = tk.Scrollbar(self,orient='horizontal')

# yscrollbar.config(command=canvas.yview)

# xscrollbar.config(command=canvas.xview)

# SaisiInfo\_frame.bind(

# "<Configure>",

# lambda e: canvas.configure(

# scrollregion=canvas.bbox("all")

# )

# )

# canvas.create\_window((0, 0), window=SaisiInfo\_frame, anchor="nw")

# canvas.configure(xscrollcommand=xscrollbar.set,yscrollcommand=yscrollbar.set)

# canvas.pack(side="left", fill="both", expand=True)

# yscrollbar.pack( side = 'right', fill = 'y')

# xscrollbar.pack( side = 'bottom', fill = 'x')

## Fonctions zoom in et out

Quelques morceaux de code sont disponibles [ici](https://stackoverflow.com/questions/41656176/tkinter-canvas-zoom-move-pan) et [ici](https://stackoverrun.com/fr/q/7073577)

* Choix des couleurs pour les différentes formes, ajouter une image dans la frame f2 en guise de légende.
* Une option pour changer de radio une fois qu’on a fini de travailler sur la première.
* Une fois qu’on aura une base de données construite, l’option d’ouvrir des radios sur lesquels on a déjà effectué les tracés
* Un bouton retour : pour annuler la dernière action.