devoir N1 python

# libraries :

* matplotlib.pyplot
* numpy
* tkinter
* os
* cv2
* sys

# 2-première tâche(task0.py) :

correct = []

lm = [0, 0]

def comression(A, lm, ecar\_type):

    l1 = []

    l11 = []

    l2 = []

    l22 = []

    l3 = []

    l33 = []

    l4 = []

    l44 = []

    for id1, i in enumerate(A):

        for id2, j in enumerate(i):

            if id2 <= int(len(A)/2-1) and id1 <= int(len(A)/2-1):

                l1.append(j)

                if lm == [0, 0]:

                    l11.append([id1, id2])

                else:

                    a = lm[0]

                    b = lm[1]

                    l11.append((id1+a, id2+b))

            if id2 > int(len(A)/2-1) and id1 <= int(len(A)/2-1):

                l2.append(j)

                if lm == [0, 0]:

                    l22.append([id1, id2])

                else:

                    l22.append((id1+lm[0], id2+lm[1]))

            if id2 <= int(len(A)/2-1) and id1 > int(len(A)/2-1):

                l3.append(j)

                if lm == [0, 0]:

                    l33.append([id1, id2])

                else:

                    l33.append((id1+lm[0], id2+lm[1]))

            if id2 > int(len(A)/2-1) and id1 > int(len(A)/2-1):

                l4.append(j)

                if lm == [0, 0]:

                    l44.append([id1, id2])

                else:

                    l44.append((id1+lm[0], id2+lm[1]))

    l = int(len(A)/2)

    mat1 = []

    while l1 != []:

        mat1.append(l1[:l])

        l1 = l1[l:]

    mat2 = []

    while l2 != []:

        mat2.append(l2[:l])

        l2 = l2[l:]

    mat3 = []

    while l3 != []:

        mat3.append(l3[:l])

        l3 = l3[l:]

    mat4 = []

    while l4 != []:

        mat4.append(l4[:l])

        l4 = l4[l:]

    if np.std(mat1) < ecar\_type or len(mat1) <= 2:

        res = [sum(idx) / len(idx) for idx in zip(\*mat1)]

        res = sum(res) / len(res)

        correct.append([l11[0][0], l11[0][1], int(len(mat1)), res])

    else:

        lm1 = [0, 0]

        lm1[0] = l11[0][0]

        lm1[1] = l11[0][1]

        comression(mat1, lm1, ecar\_type)

    if np.std(mat2) < ecar\_type or len(mat2) <= 2:

        res = [sum(idx) / len(idx) for idx in zip(\*mat2)]

        res = sum(res) / len(res)

        correct.append([l22[0][0], l22[0][1], int(len(mat2)), res])

    else:

        lm2 = [0, 0]

        lm2[0] = l22[0][0]

        lm2[1] = l22[0][1]

        comression(mat2, lm2, ecar\_type)

    if np.std(mat3) < ecar\_type or len(mat3) <= 2:

        res = [sum(idx) / len(idx) for idx in zip(\*mat3)]

        res = sum(res) / len(res)

        correct.append([l33[0][0], l33[0][1], int(len(mat3)), res])

    else:

        lm3 = [0, 0]

        lm3[0] = l33[0][0]

        lm3[1] = l33[0][1]

        comression(mat3, lm3, ecar\_type)

    if np.std(mat4) < ecar\_type or len(mat4) <= 2:

        res = [sum(idx) / len(idx) for idx in zip(\*mat4)]

        res = sum(res) / len(res)

        correct.append([l44[0][0], l44[0][1], int(len(mat4)), res])

    else:

        lm4 = [0, 0]

        lm4[0] = l44[0][0]

        lm4[1] = l44[0][1]

        comression(mat4, lm4, ecar\_type)

Donc dans le fichier task0.py, nous avons une fonction ‘ compression ‘  qui recherche des sous-matrices homogènes dans la matrice mère et ajoute dans une liste globale appelle "correct" toutes les informations sur chaque sous-matrice homogène.

Donc pour chaque matrice nous avons 4 informations :

L'indice de départ (i,j) de cette sous matrice par rapport à la matrice mère et sa longueur et enfin sa moyenne

def is\_power\_of\_two(number: int) -> bool:

    while number != 1:

        if number % 2:

            return False

        number /= 2

    return True

Et à propos de la fonction ‘is\_power\_of\_two’ qui vérifie si l'argument passé est de puissance du 2 ou no

def moyenne(l):

    s = 0

    for i in range(len(l)):

        s = s+l[i]

    s = s/len(l)

    return s

def moyen(m):

    for i in range(len(m)):

        for j in range(len(m)):

            m[i][j] = int(moyenne(m[i][j]))

    return m

Fonction ‘moyen(m)’ qui transforme votre image RGB en image noir et blanc liste 2 démentions

def getm(image, filename, ecar\_type):

    img = imgageio.imread(image)

    img = img.tolist()

    lenght\_img0 = len(img)

    lenght\_img1 = len(img[0])

    if lenght\_img1 != lenght\_img0:

        print("image does not have  the  Same Height and Width ")

    else:

        if is\_power\_of\_two(lenght\_img0) and is\_power\_of\_two(lenght\_img1):

            m = moyen(img)

            comression(m, lm, ecar\_type)

            file = open(filename, 'w')

            for i in correct:

                for j in i:

                    file.write(str(j) + "\n")

            file.close

        else:

            print("width or height of image is not power of two")

Et enfin la fonction ‘getm(image, filename, ecar\_type)’ qui utilise toutes les fonctions précédentes

obtenir d'abord la matrice de l'image à l'aide de la librairie image.io puis vérifiez si la largeur et la hauteur sont les mêmes puis vérifiez la puissance de la largeur et de la hauteur puis nous écrirons le retour de la fonction compression dans un fichier

# 3-deuxième tâche (task1.py):

def newone(filename):

    file = open(filename, 'r')

    newm = file.readlines()

    file.close

    return newm

def implemnt(filename):

    o = newone(filename)

    mylist = []

    # changing from normal list to list 2 dimensions

    for id, i in enumerate(o):

        j = i.partition('\n')

        mylist.append(float(j[0]))

    # print(mylist)

    mylist\_4 = []

    while mylist != []:

        mylist\_4.append(mylist[:4])

        mylist = mylist[4:]

    return mylist\_4

Dans ce fichier nous avons la fonction ‘implément’ qui prend un fichier en paramètres puis transforme ce fichier en list du 2 dimension

Pour chaque sous liste on a 4 informations start index(i,j) de matrice et sa longueur puis sa moyenne.

# creation matrix null

def get\_lenght(image):

    img = imgageio.imread(image)

    img = img.tolist()

    lenght\_img0 = len(img)

    return lenght\_img0

# NULL\_MATRIXE = [[0 for j in range(get\_lenght("11.png"))]

#                 for i in range(get\_lenght("11.png"))]

def create\_null\_m(image):

    my\_matrixe = np.zeros((get\_lenght(image), get\_lenght(image)))

    return my\_matrixe

Et aussi nous avons la fonction ‘create\_null\_m(image)’ qui prend une image en paramètres pour créer une matrice nulle avec la même longueur d'image

# 3-tache finale(gui.py) :

    def \_\_init\_\_(self):

        self.window = Tk()

        self.window.title("compressionApp")

        self.window.geometry("720x480")

        self.window.minsize(480, 360)

        self.window.iconbitmap("python.ico")

        self.window.config(background='#41B77F')

        # initialization des composants

        self.frame = Frame(self.window, bg='#41B77F')

        # creation des composants

        self.create\_widgets()

        # empaquetage

        self.frame.pack(expand=YES)

donc dans ce fichier il y a une implimentation de l'interface graphique :

la fonction ‘\_\_init\_\_(self)’ de la classe ‘myApp’ va insialiser les fenêtres ..

    def create\_widgets(self):

        self.create\_title()

        self.create\_subtitle("ecart type")

        self.input\_ecartype = Entry(self.frame, font=("Courrier", 40), bg='white',

                                    fg='#41B77F')

        self.input\_ecartype.pack()

        self.create\_subtitle("ur new image name")

        self.input\_newimage = Entry(self.frame, font=("Courrier", 40), bg='white',

                                    fg='#41B77F')

        self.input\_newimage.pack()

        self.create\_button()

donc dans cette initialisation, nous allons appeler la fonction ‘ create\_widgets ’ qui implémente les sous-titres et les entrées

    def create\_button(self):

        yt\_button = Button(self.frame, text="get ur image", font=(

            "Courrier", 25), bg='white', fg='#41B77F', command=self.getimage)

        yt\_button.pack(pady=25, fill=X)

et dans cette fonction (‘ create\_widgets ’) nous avons appelé la fonction ‘ create\_button ’ qui appelle la fonction ‘getimage(self)’ s'elle avait une action (click).

    def getimage(self):

        bool1 = False

        self.image = filedialog.askopenfilename(initialdir=os.getcwd(), title="select image file", filetypes=(

            ("JPG File", "\*.jpg"), ("PNG File", "\*.png"), ("ALL Files", "\*.\*")))

        print(self.image)

        self.ecarttype = self.input\_ecartype.get()

        self.newimage = self.input\_newimage.get()

        self.filename = ''.join(random.choices(string.ascii\_uppercase +

                                               string.digits, k=4))+"1.txt"

        print(self.filename)

        getm(self.image, str(

            self.filename), float(self.ecarttype))

        while (True):

            file\_exists = exists(self.filename)

            if file\_exists:

                break

        mylist\_4 = implemnt(str(self.filename))

        my\_matrixe = create\_null\_m(str(self.image))

        def getmatrixe(b):

            for i in range(int(mylist\_4[b][0]), int(mylist\_4[b][2]+int(mylist\_4[b][0]))):

                for j in range(int(mylist\_4[b][1]), int(mylist\_4[b][2]+int(mylist\_4[b][1]))):

                    my\_matrixe[i][j] = int(mylist\_4[b][3])

            b = b+1

            if b < len(mylist\_4):

                # print(b)

                getmatrixe(b)

            return my\_matrixe, True

        self.ourlistcompressed, bool1 = getmatrixe(0)

        if bool1:

            self.create\_subtitle("ur image has been created")

        self.window.destroy

        image1 = cv2.imwrite(str(self.newimage), self.ourlistcompressed)

        img = cv2.imread(self.newimage, cv2.IMREAD\_ANYCOLOR)

        file\_size = os.path.getsize(str(self.image))

        file\_size\_new = os.path.getsize(str(self.newimage))

        self.gange = abs(file\_size-file\_size\_new)

        print(self.gange)

        self.create\_subtitle("gain is "+str(self.gange)+" bytes")

        cv2.imshow("", img)

dans la fonction ’ getimage(self) ’ d'abord on va demander à l'utilisateur de donner l'image(

        self.image = filedialog.askopenfilename(initialdir=os.getcwd(), title="select image file", filetypes=(

            ("JPG File", "\*.jpg"), ("PNG File", "\*.png"), ("ALL Files", "\*.\*")))

) puis nous appellerons ‘ getm ‘ fonction de task0.py pour obtenir le fichier puis nous appellerons la fonction ’implément’ de task1.py pour obtenir la list du 2 demention à partir du fichier généré puis nous appellerons la fonction create\_null\_m de task1.py

dans ’ getimage(self) ’ on va cree puis appeler la fonction récursive ‘getmatrix(b)’ qui retourne la matrice compressée de l'image

et enfin, nous allons utiliser la bibliothèque cv2 pour afficher l'image compressée, puis on va afficher le gain (taille de l'image - taille du fichier)

# 4- les problèmes du App :

        def getmatrixe(b):

            for i in range(int(mylist\_4[b][0]), int(mylist\_4[b][2]+int(mylist\_4[b][0]))):

                for j in range(int(mylist\_4[b][1]), int(mylist\_4[b][2]+int(mylist\_4[b][1]))):

                    my\_matrixe[i][j] = int(mylist\_4[b][3])

            b = b+1

            if b < len(mylist\_4):

                # print(b)

                getmatrixe(b)

            return my\_matrixe, True

La fonction ‘getmatrixe(b)’ est une fonction récurcive donc le problème est si le fichier a une grande taille donc dans ce cas le programme s'arrêtera parce que nous avons dépasséLa limite de récursivité

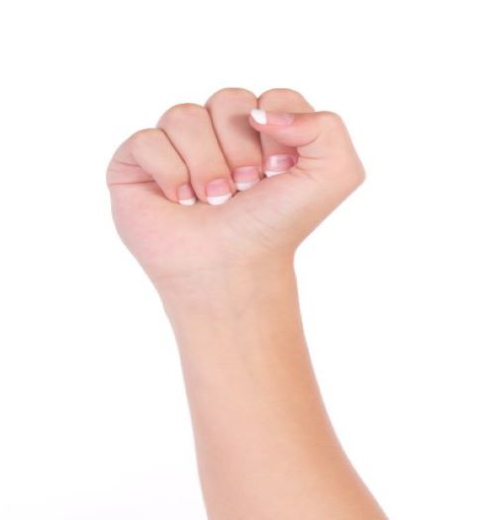
j'ai essayé de résoudre ce problème avec le sys.setrecursionlimit(30500) pour augmenter La limite de récursivité

mais en python La limite de récursivité est généralement de 1000.

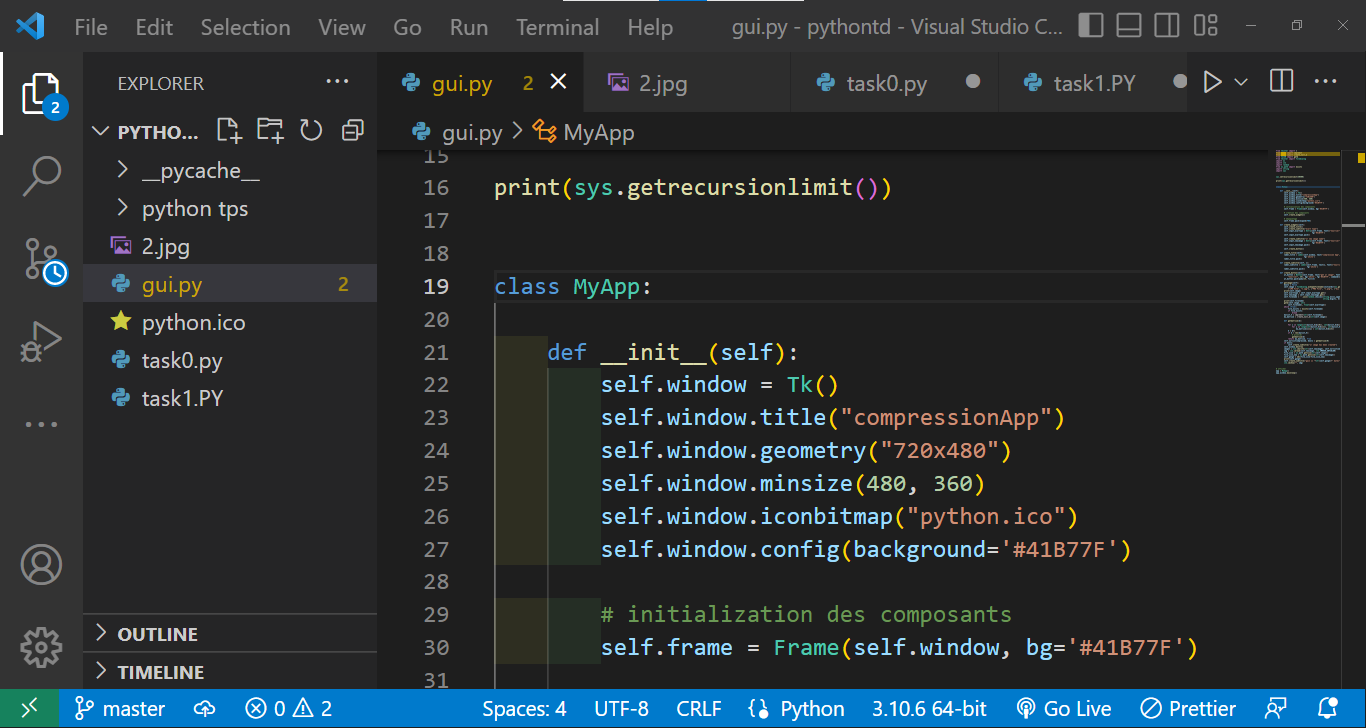
alors voici comment nous pouvons faire pour ne pas tomber dans ce problème, nous allons choisir un grande écart type et nous diminuerons lentement ce variable jusqu'à ce que nous obtenions le bon résultat

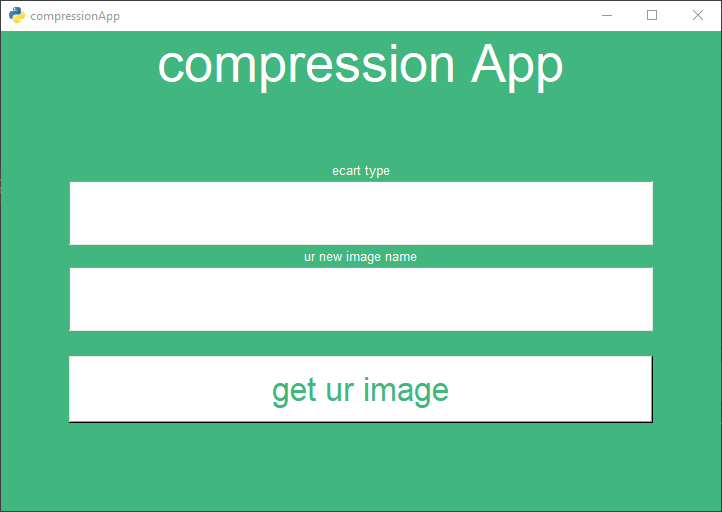
# 5-test :

On va utuliser cette image :

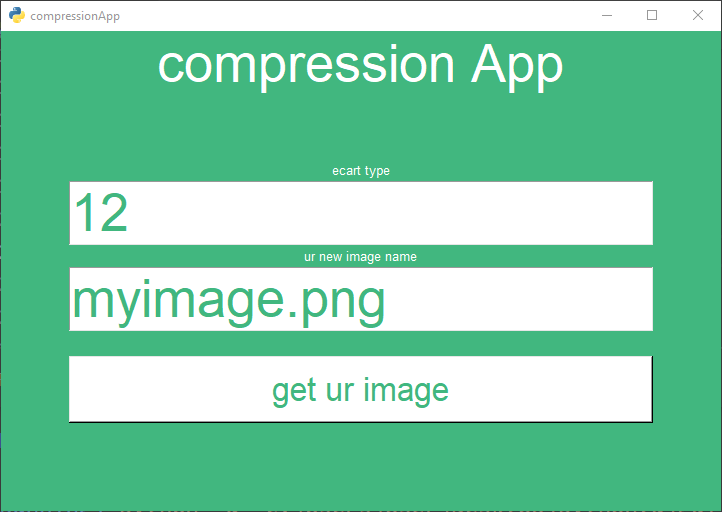


exécutons le fichier gui.py :

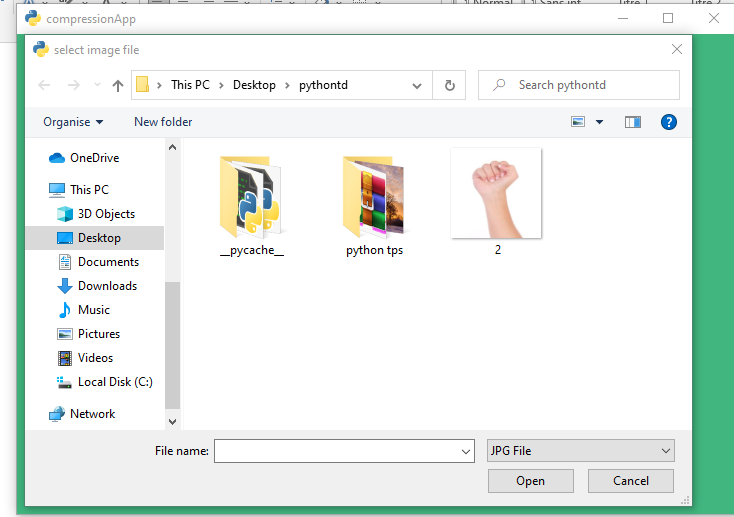




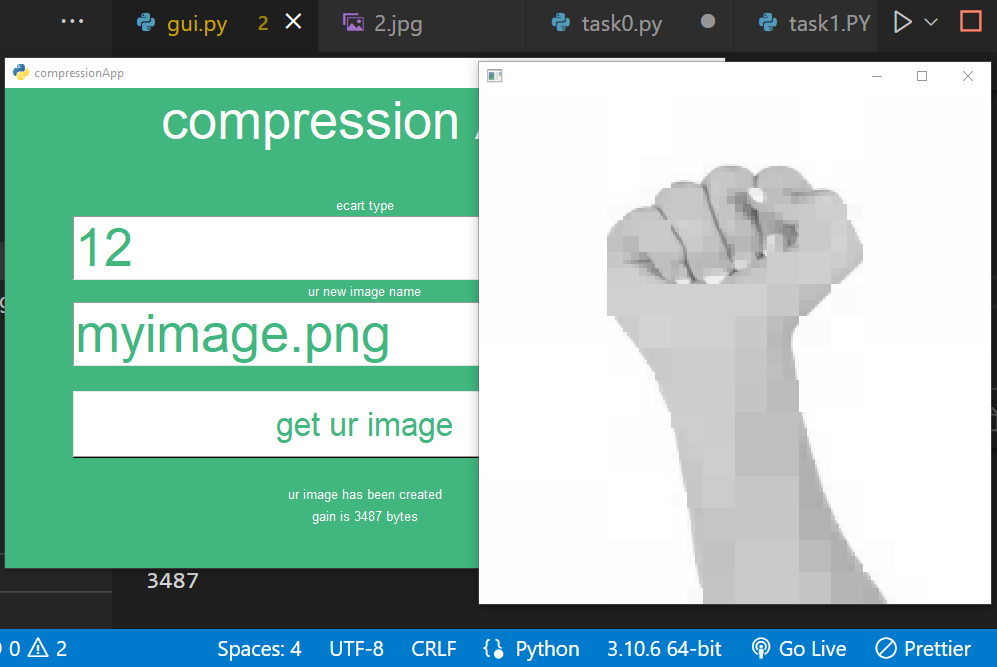
maintenant nous allons remplir les données :



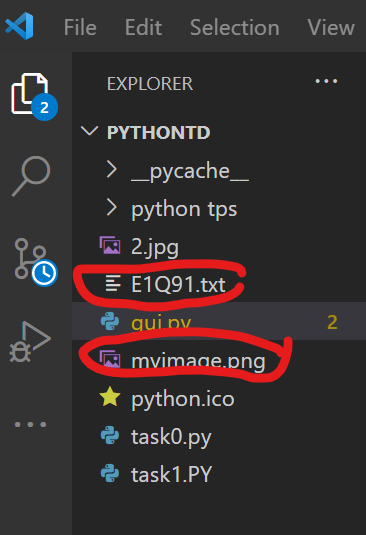
maintenant je vais cliquer sur le bouton :



maintenant je vais choisir image :

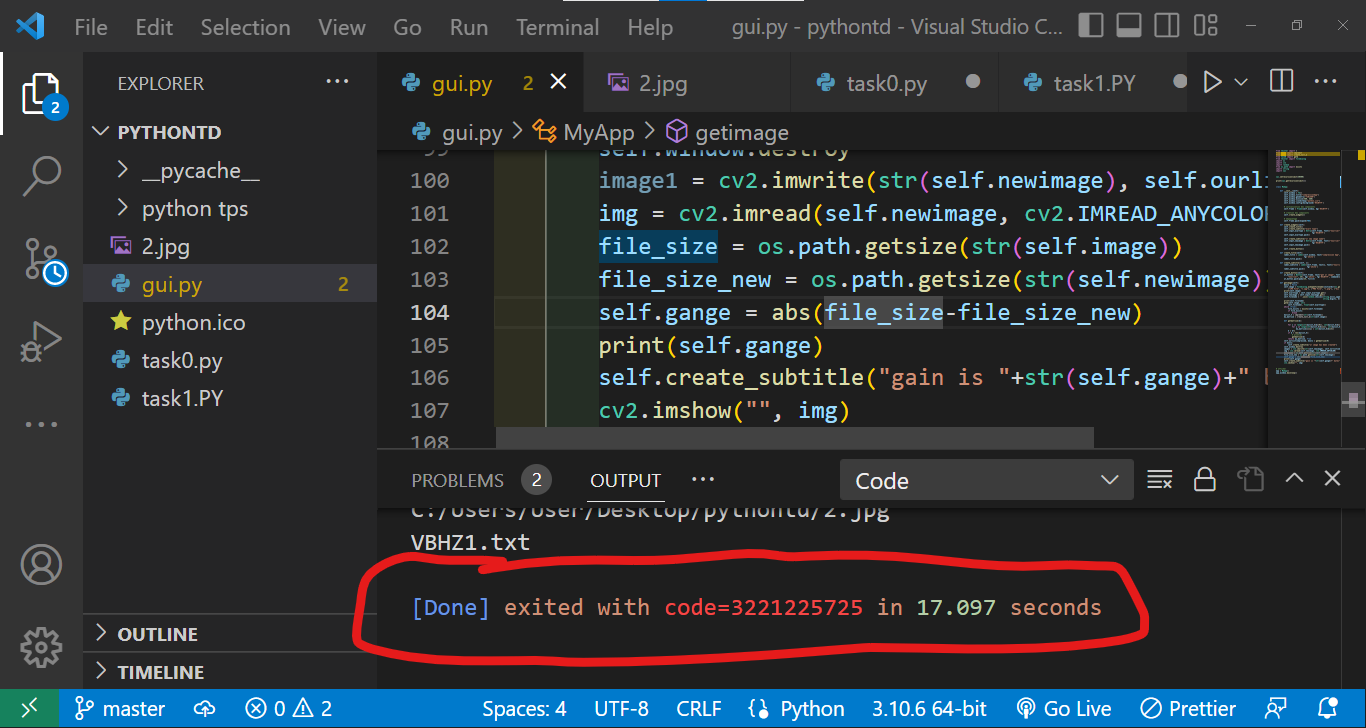


dans /src nous avons notre image compressée avec le fichier :



maintenant je vais montrer le problème que j'ai recommandé précédemment :





Le programme est quitté. A cause : RecursionError: maximum recursion depth exceeded in comparison