

**OBJECTIFS** : Ce TP est la suite du Tp11\_A. On continue à compléter le code déjà réalisé en incluant des images et des évènements. Comme dans le Tp précédent, l'évaluation de ce travail est basée sur le rendu du fichier `.py` qui sera constitué.

⇒ Pour débiter, se placer dans le répertoire utilisé dans le Tp11\_A précédent et ouvrir le fichier `tp11.py` que vous avez écrit à l'issue de cette partie A.

Néanmoins, pour être sûr de démarrer sur de bonnes bases, vous pouvez partir du code suivant, que vous pouvez copier-coller à partir de ce pdf (les indentations seront à recréer).

```
# Modules -----
from tkinter import Tk , Canvas , Label , Text , Button
from PIL import Image, ImageTk # pip install pillow
from random import randint

# Fonctions -----
def creer_fenetre() :
    fenetre = Tk()
    fenetre.title("tp11")
    return fenetre

def creer_widgets() :
    zone_graphique = Canvas(fenetre, width=1000, height=600 , bg = 'black')
    zone_graphique.grid(row = 0 , column = 0 , columnspan = 3 )
    mon_texte = Label(fenetre, text = "Entre un mot : ")
    mon_texte.grid(row = 1 , column = 0)
    champ_saisie = Text(fenetre , height = 1 , width = 14)
    champ_saisie.grid(row = 1 , column = 1)
    bouton_valider = Button(fenetre, text = "Valider" , width = 12 , command = debut)
    bouton_valider.grid(row = 1, column = 2)
    return zone_graphique, mon_texte, champ_saisie, bouton_valider

def debut():
    print('tu as cliqué sur le bouton')
    mot = champ_saisie.get("1.0", "end-1c")
    print('le texte entré est : ', mot)

# Main -----
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

fenetre.mainloop()
```

## 1. PREPARER LES IMAGES QUI SERONT UTILISEES :

Les images introduites dans la zone graphique doivent avoir le bon format et la bonne taille.

### Quel format doit-on utiliser ? :

Question format, on privilégie les formats `.png` et `.jpg` . Ces 2 formats sont aussi majoritairement utilisés sur le web.

- Le format `.png` permet de compresser une image, sans perte de données lors de la compression. Il présente l'avantage de pouvoir avoir certaines parties de l'image transparentes grâce à la présence d'un canal dit « *alpha* ».




- Le format *.jpg* compresse l'image **d'une manière plus importante**. Les fichiers ont ainsi une taille plus petite qu'avec le format *.png*. Par contre, cette compression se fait avec perte irréversible de données et ce format ne gère pas la transparence.

**La taille est-elle importante ? :**

Même avec de la transparence, les images sont incluses dans une zone rectangulaire dont la taille est donnée en pixels. Une image de 200 x 100 a une largeur (*width*) de 200 px et une hauteur (*height*) de 100 px.

La taille en octets du fichier dans lequel est enregistré l'image, devient un paramètre important lorsque l'on travaille sur le web. Cette taille est proportionnelle à la largeur et à la hauteur de l'image. Si par exemple, on divise par 2 la largeur **et** par 2 la hauteur, la taille du fichier sera à peu près divisée par 4.

On se propose dans ce paragraphe, d'utiliser le logiciel *Gimp* pour créer un fichier *.png* de l'image

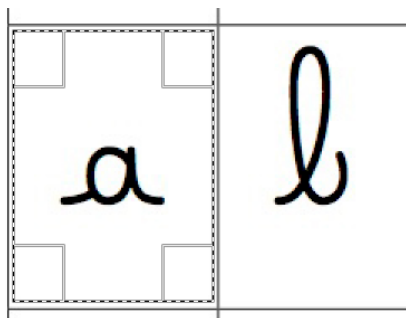
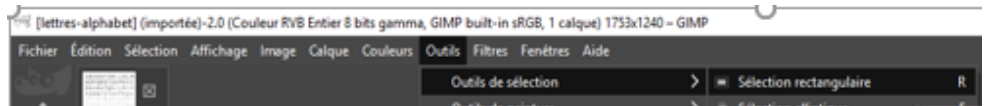
ci-contre :  . Elle sera ensuite introduite dans la zone graphique *Tkinter* de notre code.

⇒ Dans l'explorateur de fichier windows, allez dans votre répertoire de travail et avec un click droit sur le fichier ***lettres-alphabet.jpg***, faites : *Ouvrir avec ... Gimp*

⇒ Avec la molette de la souris, zoomer autour de la lettre a minuscule :



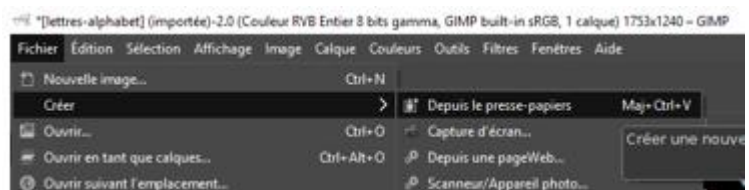
⇒ Dans *Outils*, utiliser *Outils de Sélection / Sélection rectangulaire* pour sélectionner la partie de l'image à l'intérieur du cadre de cette lettre ***a*** :



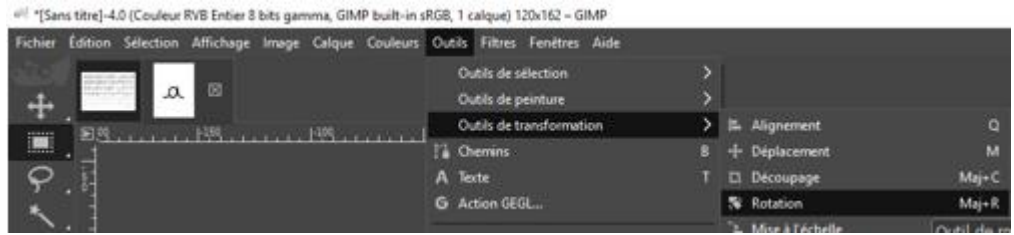
⇒ Copier cette sélection dans le presse-papier par un *Ctrl C*

⇒ Créer une autre image en cliquant :

*Fichier / Créer / Depuis le presse-papiers*



⇒ Cliquer sur : *Outils / Outils de transformation / Rotation* :

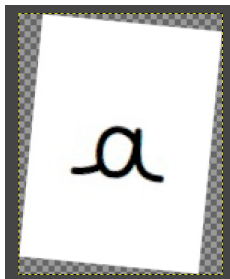


⇒ Tourner légèrement l'image en jouant sur le curseur et cliquer sur : *Rotation* pour finaliser :

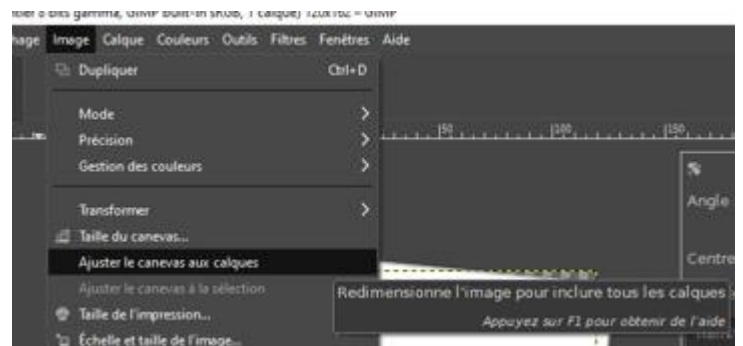


⇒ Cliquer sur :

*Image / Ajuster le canevas aux calques*

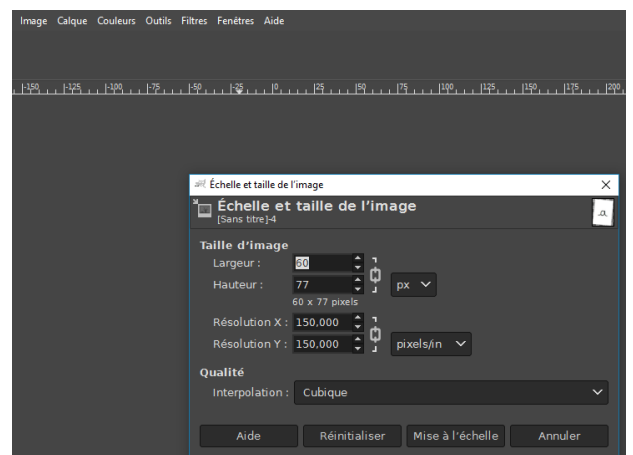


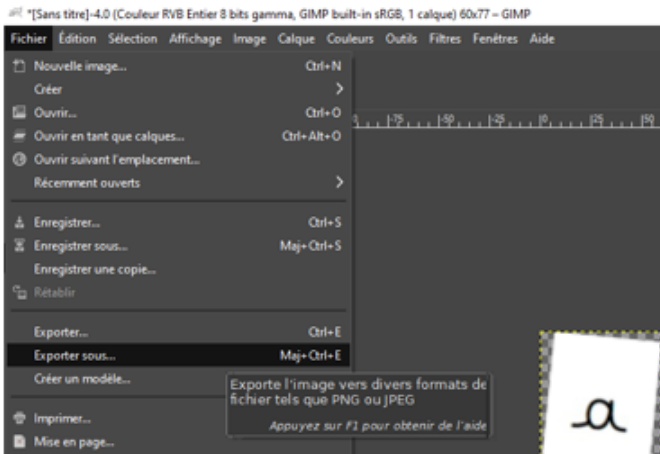
afin de pouvoir obtenir une image rectangulaire qui inclue le cadre blanc légèrement tourné et le fond transparent.



⇒ Cliquer sur : *Image / Echelle et taille de l'image*

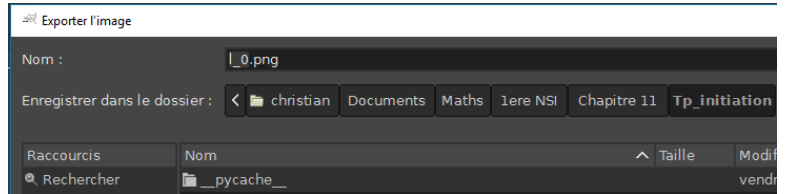
et réduire la *Largeur* à 60 px . La *Hauteur* change alors automatiquement pour ne pas déformer l'image. Cliquer sur *Mise à l'échelle* pour finaliser l'opération.





⇒ Cliquer sur : *Fichier / Exporter sous ...*

Afin d'enregistrer cette image sous le nom : *l\_0.png*



⇒ Vous avez à présent dans votre répertoire de travail cette image *l\_0.png* qui s'est rajoutée aux images nommées *l\_1.png*, *l\_2.png*, ..... jusqu'à *l\_26.png*, qui reprennent chacune une lettre de l'alphabet minuscule au format *.png* avec une taille de 60 x 80 px. La dernière image *l\_26.png* correspond à une image blanche de 60 x 80 px.

### 2. Que va-t-on faire dans ce Tp11\_B ? :

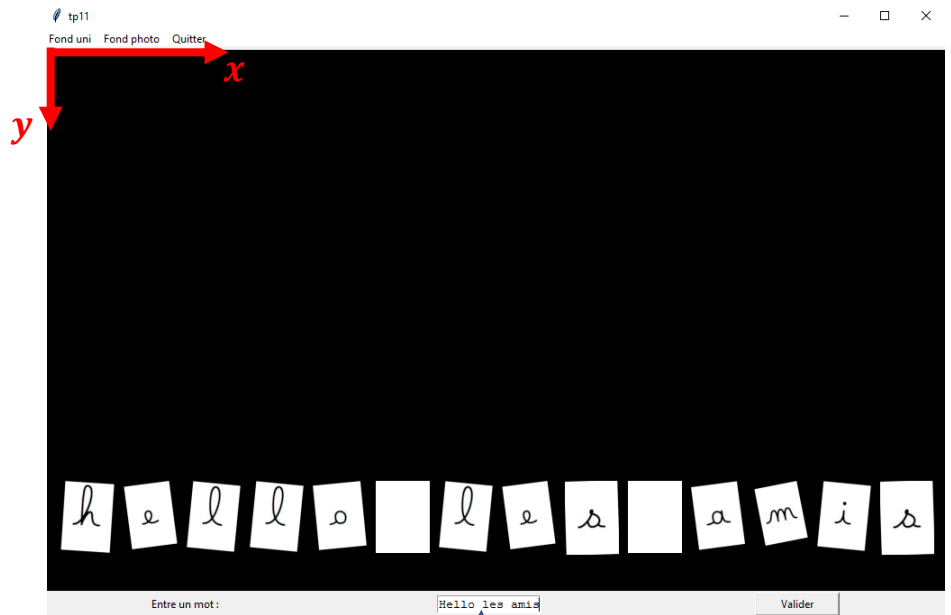
On se propose dans ce Tp11\_B de compléter le code réalisé dans le Tp11\_A, afin de,

- après saisie d'un texte de 14 lettres au maximum dans le champs de saisie,
- après click sur le bouton *Valider*,

pouvoir afficher le même texte avec les lettres images de 60x80 px vues précédemment.

Exemple sur la figure ci-contre après avoir saisi

« *Hello les amis* »



### 3. Comment fait-on pour insérer 1 image dans un Canvas ? :

⇒ Copier les 2 lignes encadrées ci-dessous, dans le programme principal et exécuter.

```
# Main -----
fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

pic = ImageTk.PhotoImage(Image.open("l_0.png"), master = fenetre)
num = zone_graphique.create_image(300, 200, anchor = "nw", image = pic)

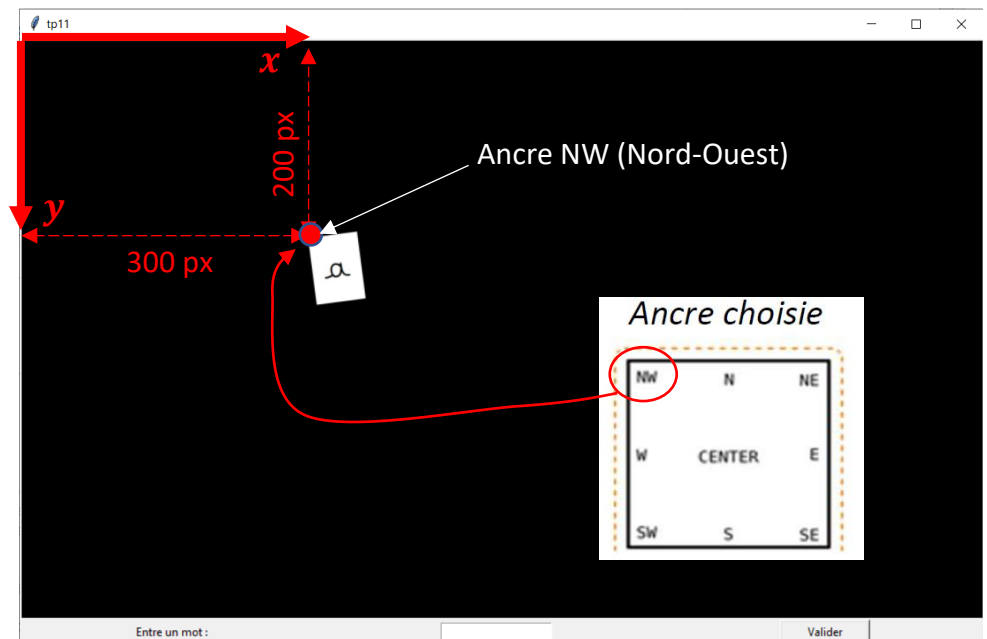
fenetre.mainloop()
```

La signification de chacune de ces 2 lignes est la suivante :

- Sur la ligne `pic = ImageTk.PhotoImage(Image.open("l_0.png") , master = fenetre)` , on crée un objet image dans l'objet '*fenetre*' de *Tkinter*, à partir du fichier '*l\_0.png*'. Cet objet est stocké dans une variable qui s'appelle ici *pic* .
- Sur la ligne `num = zone_graphique.create_image(300 , 200 , anchor = "nw" , image = pic)` on **insère** cet objet dans le *canvas* qui est stocké dans la variable *zone\_graphique*. Pour cela, on utilise la méthode *create\_image()*.

A l'intérieur de ce *canvas*, cette image est repérée ici avec un numéro qui est stocké dans la variable *num*.

Pour **positionner** cette image dans le *canvas*, on précise les coordonnées de son **anc**re dans un repère donné ci-contre :



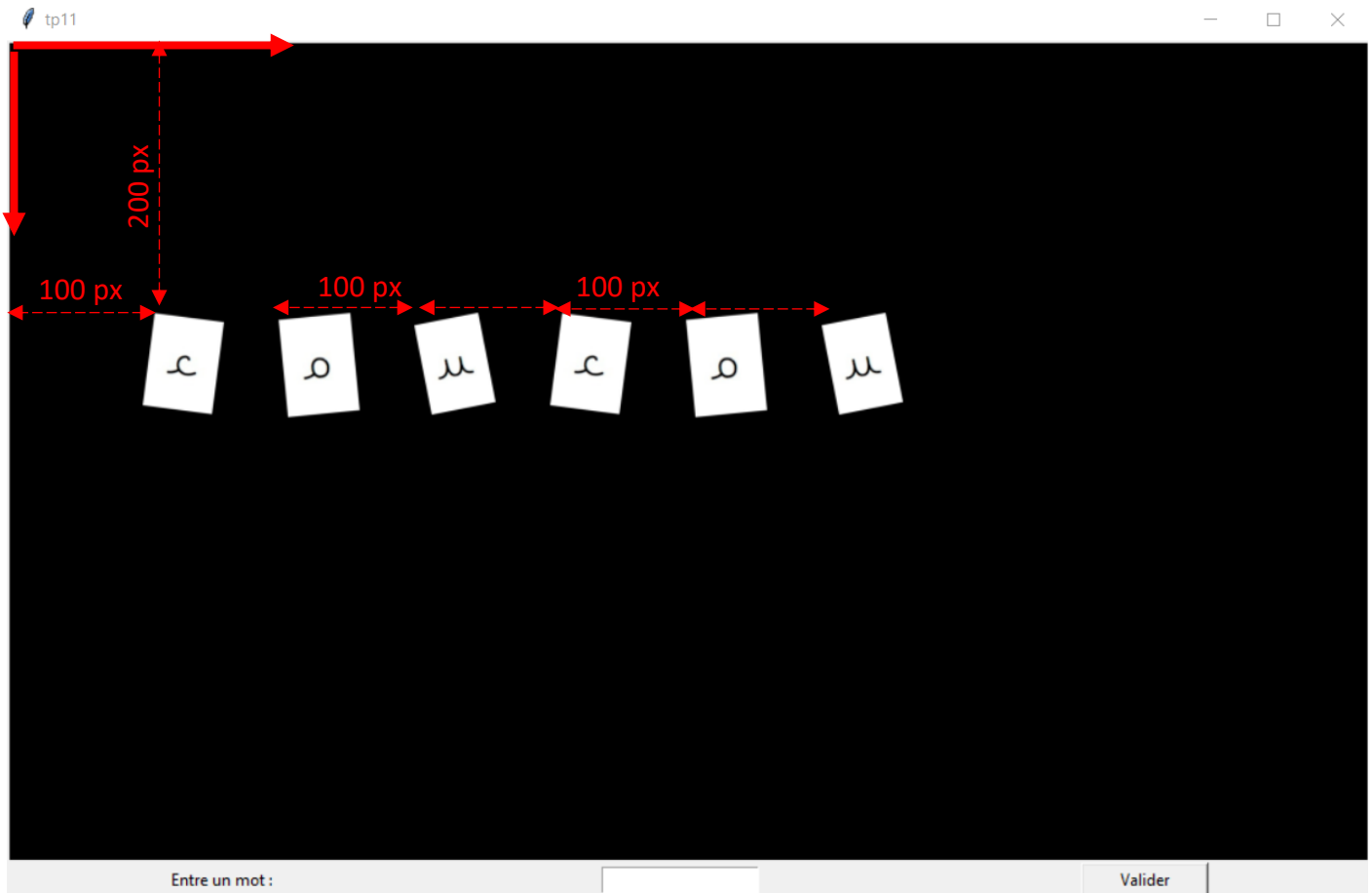
⇒ Dans le shell, exécuter la commande `>>> pic` . Qu'obtient-on ?

⇒ Dans le shell, exécuter la commande `>>> num` . Qu'obtient-on ?

⇒ Peut-on dire que l'image est repérée dans le *canvas* avec un simple numéro ?

4. On continue en insérant plusieurs images et en les stockant dans un dictionnaire :

⇒ Sachant que les fichiers images 'l\_i.png' ont une largeur de 60px, modifier et compléter le programme principal pour obtenir à l'exécution :



Un code possible est :

```
# Main -----
fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

pic1 = ImageTk.PhotoImage(Image.open("l_2.png"), master = fenetre)
pic2 = ImageTk.PhotoImage(Image.open("l_14.png"), master = fenetre)
pic3 = ImageTk.PhotoImage(Image.open("l_20.png"), master = fenetre)
pic4 = ImageTk.PhotoImage(Image.open("l_2.png"), master = fenetre)
pic5 = ImageTk.PhotoImage(Image.open("l_14.png"), master = fenetre)
pic6 = ImageTk.PhotoImage(Image.open("l_20.png"), master = fenetre)
num = zone_graphique.create_image(100, 200, anchor = "nw", image = pic1)
num = zone_graphique.create_image(200, 200, anchor = "nw", image = pic2)
num = zone_graphique.create_image(300, 200, anchor = "nw", image = pic3)
num = zone_graphique.create_image(400, 200, anchor = "nw", image = pic4)
num = zone_graphique.create_image(500, 200, anchor = "nw", image = pic5)
num = zone_graphique.create_image(600, 200, anchor = "nw", image = pic6)

fenetre.mainloop()
```

C'est long .... plus simple de stocker tous les objets images *pic*.. dans un dictionnaire nommé *pic* par exemple, pour pouvoir écrire ensuite :

```
# Main -----
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

pic = {}
i = 0
for c in "abcdefghijklmnopqrstuvw " :
    fichier = "l_"+str(i)+".png"
    i = i + 1
    pic[c] = ImageTk.PhotoImage(Image.open(fichier) , master = fenetre)

zone_graphique.create_image(100 , 200 , anchor = "nw" ,image = pic["c"])
zone_graphique.create_image(200 , 200 , anchor = "nw" ,image = pic["o"])
zone_graphique.create_image(300 , 200 , anchor = "nw" ,image = pic["u"])
zone_graphique.create_image(400 , 200 , anchor = "nw" ,image = pic["c"])
zone_graphique.create_image(500 , 200 , anchor = "nw" ,image = pic["o"])
zone_graphique.create_image(600 , 200 , anchor = "nw" ,image = pic["u"])

fenetre.mainloop()
```

⇒ Exécuter ce code.

⇒ Dans le shell, exécuter ensuite l'instruction `>>> pic["a"]` , puis l'instruction `>>> pic["b"]` .

Quels sont les contenus stockés dans ce dictionnaire nommé *pic* ?

⇒ Quelle image s'affichera si on insère dans le canvas `pic[" "]` ?

5. On améliore le script précédent pour le simplifier :

Pour pouvoir travailler dans un programme principal plus aéré, on définit la fonction *creation\_dictionnaire\_pic()* que l'on appelle dans le programme principal. On en profite pour éviter d'appeler 6 fois la méthode *create\_image()* , ce qui nous conduit au code ci-dessous :

```
def creation_dictionnaire_pic():
    i = 0
    for c in "abcdefghijklmnopqrstuvw " :
        fichier = "l_"+str(i)+".png"
        i = i + 1
        pic[c] = ImageTk.PhotoImage(Image.open(fichier) , master = fenetre)

# Main -----
pic = {}
picsDansCanvas = []

fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

creation_dictionnaire_pic()
i = 0
for lettre in "coucou" :
    num = zone_graphique.create_image(100 + 100*i , 200 , anchor = "nw" ,image = pic[lettre])
    picsDansCanvas.append(num)
    i = i + 1

fenetre.mainloop()
```

Dans ce code, les numéros des images insérées dans le *canvas* sont stockés dans une liste nommée *picsDansCanvas[]* . Cette liste étant déclarée dans le programme principal, elle est accessible en lecture et écriture **dans tout le script**, même dans les fonctions. Il en est de même du dictionnaire nommé *pic*

⇒ Saisir et exécuter ce code.

⇒ Modifier le code pour afficher dans le *canvas* la phrase : « *plus velo* »

⇒ Dans le shell, exécuter alors l'instruction : `>>> picsDansCanvas` pour constater qu'il s'agit bien d'une liste qui ne contient **que les numéros des images qui ont été insérées dans le canvas**.

6. On modifie encore ce code, pour afficher les images uniquement après un click sur le bouton *Valider* :

⇒ Modifier encore le code en complétant celui donné ci-dessous. Il permet d'afficher dans le *canvas*, le mot écrit dans le champ de saisie, uniquement **après click sur le bouton valider** :

```
def debut():
    mot = champ_saisie.get("1.0", "end-1c")

def creation_dictionnaire_pic():
    i = 0
    for c in "abcdefghijklmnopqrstuvwxy " :
        fichier = "\_"+str(i)+".png"
        i = i + 1
        pic[c] = ImageTk.PhotoImage(Image.open(fichier) , master = fenetre)

# Main -----
pic = {}
picsDansCanvas = []

fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

creation_dictionnaire_pic()

fenetre.mainloop()
```

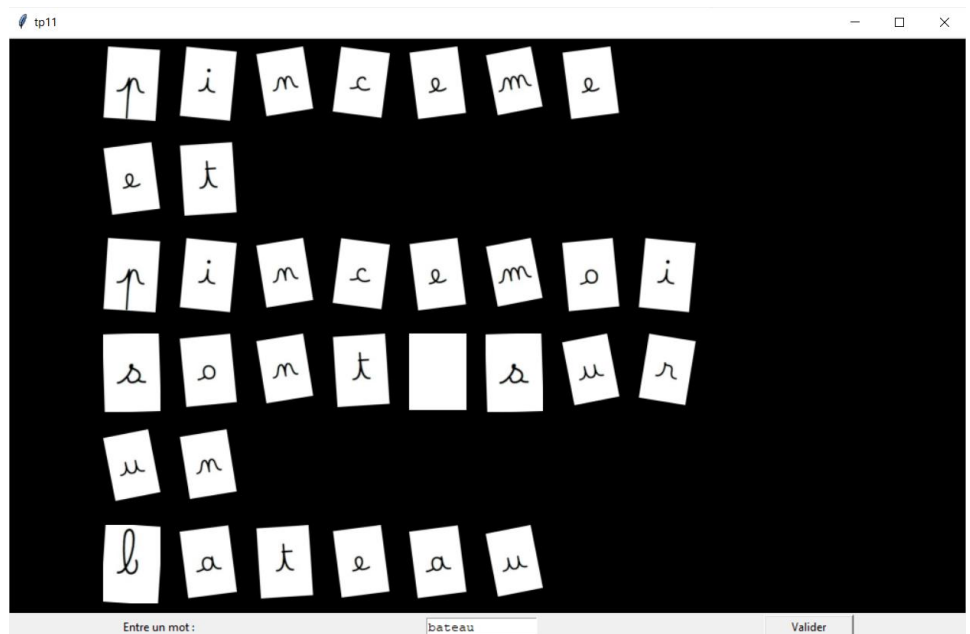


7. On modifie encore le code, pour pouvoir afficher plusieurs mots sur la hauteur.

⇒ La hauteur des fichiers images est de 77 px environ. La hauteur du canvas est de 600 px. On peut ainsi afficher sur la hauteur totale du canvas au moins 6 mots différents .

En prenant modèle sur le code incomplet ci-dessous, compléter la fonction *debut()* afin que l'utilisateur puisse afficher 6 mots sur la hauteur du *canvas*. En profiter pour réduire l'espacement entre les lettres à 80 px :

⇒ Utiliser ce code pour obtenir à l'écran :



On ajoute cette instruction afin que la variable *n* définie dans le programme principal, soit accessible en écriture dans cette fonction. On fait cela ici, car avec Tkinter, on ne peut pas mettre d'argument à une fonction liée à un événement.

```
def debut():
    global n
```

```
n = n + 1
```

```
# Main -----
pic = {}
picsDansCanvas = []
n = 0
```

On définit et initialise une variable *n* dans le programme principal

```
fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

creation_dictionnaire_pic()

fenetre.mainloop()
```

⇒ Après avoir affiché dans le canvas « *pinceme et pince moi sont sur un bateau* », exécuter dans le shell l'instruction : `>>> picsDansCanvas` . Quelle est la taille de cette liste ?

8. On complète pour créer un évènement clavier qui permet de retirer une image du canvas :

On se propose de créer un évènement lié à l'appui sur la touche ↓ du clavier (touche *Down*). Cet évènement devra retirer du *canvas*, la dernière image créée.

Dans Tkinter, les évènements « clavier » se définissent en utilisant la méthode *bind()* sur l'objet *fenetre*.

On obtient ainsi le code suivant :

```
def efface_image(event):
    if (picsDansCanvas != []):
        # on n'affiche plus la dernière image de picsDansCanvas[] dans le canvas
        num = picsDansCanvas[-1]
        zone_graphique.delete(num)
        # on supprime cette image de la liste picsDansCanvas[]
        del(picsDansCanvas[-1])
```

On doit mettre le mot clé *event* en argument de la fonction appelée. Pas le droit de mettre d'autres arguments

```
# Main -----
pic = {}
picsDansCanvas = []
n = 0

fenetre = creer_fenetre()
zone_graphique, mon_texte, champ_saisie, bouton_valider = creer_widgets()

creation_dictionnaire_pic()

fenetre.bind("<Down>", efface_image)

fenetre.mainloop()
```

On crée dans le programme principal un évènement lié à l'appui sur la touche *Down* du clavier qui sera suivi de l'exécution de la fonction *efface\_image()*

⇒ Exécuter le code ci-contre et vérifier son bon fonctionnement

9. Créer un évènement clavier qui permet de retirer toutes les images qui ont été insérées dans le *canvas* :

⇒ Compléter votre code en créant un évènement lié à l'appui de la touche ↑ (touche *Up*) qui permet en une fois, de retirer la totalité des images.

Il s'agit ici de créer l'évènement et la fonction appelée que l'on nommera *efface\_tout()* :

```
def efface_tout(event):
    global n
```

```
del(picsDansCanvas[:])
n = 0
```

Pour vider une liste *l*, on utilise la syntaxe *del(l[:])*

La valeur de *n* doit être initialisée à 0 pour que le texte saisi ensuite soit affiché à partir du haut du *canvas*.

----- FIN de cette 2<sup>ème</sup> partie -----

**DOCUMENT A RENDRE :**

Cette 2<sup>ème</sup> partie vous a permis d'insérer des images dans le *canvas* et de les supprimer. Elle vous a permis aussi de créer des évènements claviers.

Transférer le fichier *tp11.py* par l'intermédiaire de l'onglet transfert du site <https://nsibranly.fr> en utilisant le code : **tp11** .