Informatique 2M

A. Ridard

Interaction avec l'utilisateur

Gymnase Yverdon

canton de vaud

Table des matières

La plupart des programmes que nous utilisons sont *interactifs*. L'utilisateur peut alors influencer le comportement de ces programmes au moyen d'une **interface utilisateur** qui peut être **textuelle** ou **graphique**.

Mais avant d'étudier ces interfaces, intéressons-nous aux manipulations de fichiers.

I. Lecture et écriture de fichiers

Les fichiers permettent à un programme de recevoir des entrées et de produire des sorties.

Avant d'être utilisé, un fichier doit être ouvert avec **open** prenant deux paramètres :
- le fichier
- le mode d'utilisation

Pour ouvrir en écriture et lecture en conservant le contenu, on procède comme suit :

```
import os

# on commence par définir le répertoire courant
os.chdir("cheminRepertoireCourant")

# le répertoire courant étant défini, le nom du fichier suffit
f = open("monFichier.txt", "a+")
```

Donnons ici quelques modes d'utilisation :
- "**r**" : lecture seule
- "**w**" : écriture seule, le fichier est vidé de son contenu
- "**w+**" : écriture et lecture, le fichier est vidé de son contenu
- "**a**" : écriture seule, le fichier conserve son contenu
- "**a+**" : écriture et lecture, le fichier conserve son contenu

Pour manipuler le fichier stocké dans la variable f[1], on dispose de méthodes :
- **f.read(n)** : renvoie au plus n caractères, ou tout le contenu si n est absent
- **f.readline()** : renvoie la ligne[2] courante[3] du fichier
- **f.readlines()** : renvoie le tableau de toutes les lignes du fichier
- **f.write(s)** : écrit la chaîne **s** dans le fichier
- **f.writelines(t)** : écrit le tableau des lignes[4] **t** dans le fichier

On peut aussi écrire avec la fonction **print()** en passant le paramètre **file=f**

```
f = open("monFichier.txt", "w")
print("Hello World !", file=f)
```

Écrire un programme permettant à l'utilisateur de compléter le formulaire[a].

Plus précisément, pour chaque champ du formulaire, le programme doit :
- lire sur le fichier l'intitulé, et inviter l'utilisateur à saisir sa réponse
- écrire sur le fichier cette réponse, puis passer au champ suivant

A la fin de la saisie, le programme doit afficher le contenu du fichier.

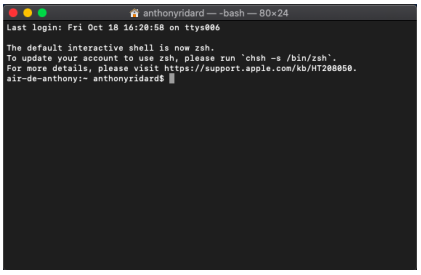
a. Le fichier *formulaire.txt* est disponible sur Moodle

[1]. f = **open**("repertoireCourant/monFichier.txt", "a+")
[2]. Les caractères de retour à la ligne sont convertis en « \n »
[3]. Un curseur interne indique où l'on se trouve dans le fichier
[4]. Les chaînes du tableau doivent avoir le caractère de fin de ligne

II. Interface textuelle

1. Terminal

Le terminal (ou console) est une interface textuelle qui permet à l'utilisateur d'interagir avec le système d'exploitation.



Elle se résume à une invite de commandes dans laquelle l'utilisateur peut écrire des commandes spécifiques correspondant à des programmes qui s'exécutent sur l'ordinateur, puis rendent la main à l'utilisateur qui peut alors saisir de nouvelles commandes. C'est l'une des interfaces historiques, utilisées avant que les ordinateurs ne soient pourvus de capacités graphiques avancées.

```
air-de-anthony:~ anthonyridard$
```

Expérimentons quelques commandes.

« pwd » fournit le répertoire courant

```
air-de-anthony:~ anthonyridard$ pwd
/Users/anthonyridard
```

« ls » permet de lister le contenu d'un répertoire

```
air-de-anthony:~ anthonyridard$ ls
Applications                    Music
Desktop                         OneDrive
Documents                       OneDrive - Education Vaud
Downloads                       Pictures
Library                         Public
Movies                          bin
```

« cd » permet de changer de répertoire

```
air-de-anthony:~ anthonyridard$ cd Desktop
air-de-anthony:Desktop anthonyridard$ pwd
/Users/anthonyridard/Desktop
```

```
air-de-anthony:Desktop anthonyridard$ cd ..
air-de-anthony:~ anthonyridard$ pwd
/Users/anthonyridard
```

```
air-de-anthony:~ anthonyridard$ cd Desktop/Annexes
air-de-anthony:Annexes anthonyridard$ pwd
/Users/anthonyridard/Desktop/Annexes
```

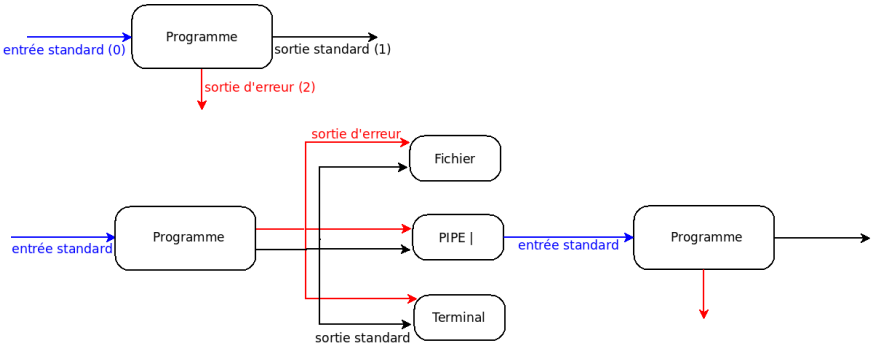
L'*option* « -l » de la commande ls permet d'afficher le détail

```
air-de-anthony:Annexes anthonyridard$ ls -l
total 20440
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
```

« * » permet de désigner tous les fichiers

```
air-de-anthony:Annexes anthonyridard$ ls -l *
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
```

Pour aller plus loin, faisons le point sur les entrées et les sorties d'un programme.



« > » permet de rediriger la sortie standard vers un fichier

```
air-de-anthony:Annexes anthonyridard$ ls -l * > liste.txt
air-de-anthony:Annexes anthonyridard$ ls
article1.txt    bonjour.py      joconde.png     liste.txt
```

« cat » permet d'afficher le contenu d'un fichier

```
air-de-anthony:Annexes anthonyridard$ cat liste.txt
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
```

« | » permet de rediriger la sortie standard vers la commande « sort »

```
air-de-anthony:Annexes anthonyridard$ ls -l * | sort -k 5 -n
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-r--r--  1 anthonyridard  staff       211 18 oct 15:58 liste.txt
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
```

« sort -k 5 -n » permet de trier selon le 5e champ qui est numérisé

```
air-de-anthony:Annexes anthonyridard$ ls -l * | sort -k 5 -n > liste.txt
air-de-anthony:Annexes anthonyridard$ cat liste.txt
-rw-r--r--  1 anthonyridard  staff         0 18 oct 16:00 liste.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
```

« > » écrase le contenu du fichier

Avez-vous remarqué le 0 octet du fichier liste.txt ?

L'option « -r » permet de trier dans l'ordre décroissant

```
air-de-anthony:Annexes anthonyridard$ ls -l * | sort -k 5 -n -r
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-r--r--  1 anthonyridard  staff       279 18 oct 16:00 liste.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
```

« > > » permet de rediriger la sortie standard sans écraser le contenu

```
air-de-anthony:Annexes anthonyridard$ ls -l * | sort -k 5 -n -r >> liste.txt
air-de-anthony:Annexes anthonyridard$ cat liste.txt
-rw-r--r--  1 anthonyridard  staff         0 18 oct 16:00 liste.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
-rw-------@ 1 anthonyridard  staff  10455194 13 déc  2023 joconde.png
-rw-------@ 1 anthonyridard  staff       720 18 oct 15:03 article1.txt
-rw-r--r--  1 anthonyridard  staff       279 18 oct 16:00 liste.txt
-rw-r--r--@ 1 anthonyridard  staff       101 18 oct 14:36 bonjour.py
```

« 2 > » permet de rediriger la sortie d'erreur vers un fichier

```
air-de-anthony:Annexes anthonyridard$ lss
-bash: lss: command not found
```

```
air-de-anthony:Annexes anthonyridard$ lss 2> erreur.txt
air-de-anthony:Annexes anthonyridard$ ls
article1.txt    bonjour.py      erreur.txt      joconde.png     liste.txt
air-de-anthony:Annexes anthonyridard$ cat erreur.txt
-bash: lss: command not found
```

« 2 > > » permet de rediriger la sortie d'erreur sans écraser le contenu

```
air-de-anthony:Annexes anthonyridard$ ls --l
ls: illegal option -- -
usage: ls [-@ABCFGHLOPRSTUWabcdefghiklmnopqrstuwx1%] [file ...]
```

```
air-de-anthony:Annexes anthonyridard$ ls --l 2>> erreur.txt
air-de-anthony:Annexes anthonyridard$ cat erreur.txt
-bash: lss: command not found
ls: illegal option -- -
usage: ls [-@ABCFGHLOPRSTUWabcdefghiklmnopqrstuwx1%] [file ...]
```

2. Bibliothèque sys

Le terminal peut exécuter un script python, et la bibliothèque **sys** permet d'accéder :

- aux arguments passés sur la ligne de commande
- aux fichiers d'entrée standard, de sortie standard et de sortie d'erreur

Arguments passés sur la ligne de commande

```python
import sys

nom = "à toi"
if len(sys.argv) == 2 :
    nom = sys.argv[1]

print("Bonjour", nom, "!")
```

```
air-de-anthony:2M_Diapos2 anthonyridard$ python3 argv.py
Bonjour à toi !
air-de-anthony:2M_Diapos2 anthonyridard$ python3 argv.py Anthony
Bonjour Anthony !
air-de-anthony:2M_Diapos2 anthonyridard$ python3 argv.py Anthony Ridard
Bonjour à toi !
```

Fichier d'entrée standard et de sortie standard

```python
import sys

print("Comment t'appelles-tu ?")
txt = sys.stdin.readline()
sys.stdout.write("Bonjour " + txt)
```

```
air-de-anthony:2M_Diapos2 anthonyridard$ python3 std.py
Comment t'appelles-tu ?
Anthony Ridard
Bonjour Anthony Ridard
```

- La méthode readline() sur sys.stdin se comporte comme la fonction input()
- La méthode write() sur sys.stdout se comporte comme la fonction print()

On savait déjà faire

```
print("Comment t'appelles-tu ?")
txt = input()
print("Bonjour " + txt)
```

```
air-de-anthony:2M_Diapos2 anthonyridard$ python3 std2.py
Comment t'appelles-tu ?
Anthony Ridard
Bonjour Anthony Ridard
```

Écrire un programme permettant de réaliser l'affichage suivant.

```
air-de-anthony:2M_Diapos2 anthonyridard$ python3 ex.py Ridard Anthony 45
Nom : Ridard
Prénom : Anthony
Age : 45
```

III. Interface graphique

1. Bibliothèque tkinter

La bibliothèque **tkinter** (**t**ool **k**it **inter**face) permet de concevoir des interfaces graphiques « portables » c'est à dire qui fonctionnent quelque soit le système d'exploitation.

Plus précisément, elle propose des composants graphiques (widgets) :

- Statiques
  - Étiquette (**Label**)
  - Zone graphique (**Canvas**)
- Dynamiques
  - Bouton (**Button**)
  - Zone de saisie (**Entry**) et chaîne dynamique (**StringVar**)
  - ...

Importation de la bibliothèque

```
import tkinter as tk
```

Syntaxe POO

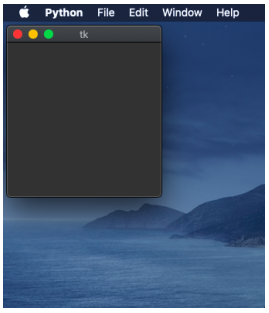
La Programmation Orientée Objet (POO) est un paradigme qui définit des classes possédant des attributs (propriétés) et des méthodes (comportements). Ces classes peuvent instancier (créer) des objets possédant alors les attributs et les méthodes de la classe.

Les trois choses à savoir ici sont :

- création d'un objet : **Classe(paramètres)**
- accès à un attribut : **objet.attribut**
- utilisation d'une méthode : **objet.méthode(paramètres)**

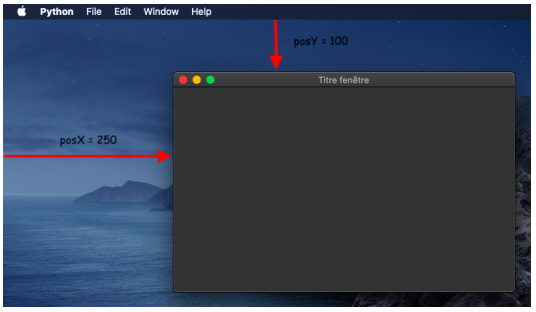
Fenêtre principale

```
fenetre = tk.Tk()
fenetre.mainloop()
```



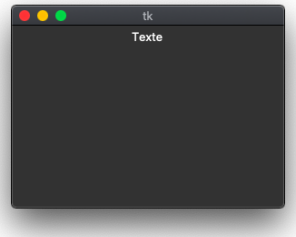
Personnalisation de la fenêtre principale

```
fenetre = tk.Tk()

fenetre.title('Titre fenêtre')
fenetre.geometry('500x300+250+100') # LARGEURxHAUTEUR+posX+posY

fenetre.mainloop()
```



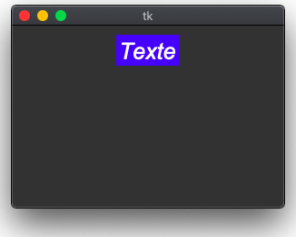
Étiquette

```
fenetre = tk.Tk()
fenetre.geometry('300x200')

etiquette = tk.Label(fenetre, text='Texte')
etiquette.pack()

fenetre.mainloop()
```



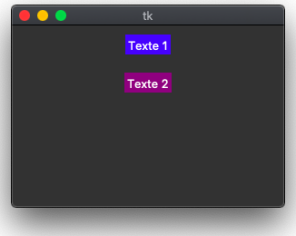
Personnalisation d'une étiquette

```
fenetre = tk.Tk()
fenetre.geometry('300x200')

etiquette = tk.Label(fenetre, text='Texte', bg='blue', fg='white', font=('Arial', 25, 'italic'))
etiquette.pack(padx=10, pady=10)

fenetre.mainloop()
```



Disposition avec la méthode pack()

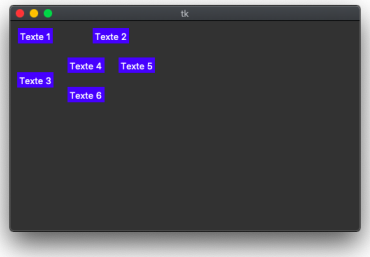
```
fenetre = tk.Tk()
fenetre.geometry('300x200')

etiquette1 = tk.Label(fenetre, text='Texte 1', bg='blue', fg='white')
etiquette2 = tk.Label(fenetre, text='Texte 2', bg='purple', fg='white')

etiquette1.pack(padx=10, pady=10)
etiquette2.pack(padx=10, pady=10)

fenetre.mainloop()
```



A partir d'ici, je vous présenterai éventuellement qu'une partie du code...

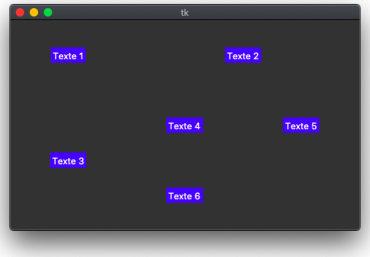
Disposition avec la méthode grid()

```
etiquette1.grid(row=0, column=0, padx=10, pady=10)
etiquette2.grid(row=0, column=1, columnspan=2, padx=10, pady=10)
etiquette3.grid(row=1, column=0, rowspan= 2, padx=10, pady=10)
etiquette4.grid(row=1, column=1, padx=10, pady=10)
etiquette5.grid(row=1, column=2, padx=10, pady=10)
etiquette6.grid(row=2, column=1, padx=10, pady=10)
```



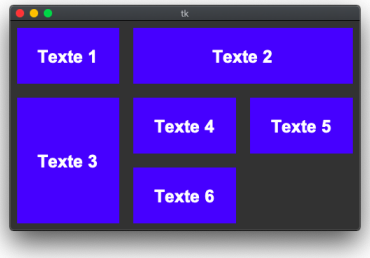
Centrage avec les méthodes grid_rowconfigure() et grid_columnconfigure()

```
fenetre = tk.Tk()
fenetre.geometry('500x300')

fenetre.grid_rowconfigure(0, weight=1)
fenetre.grid_rowconfigure(1, weight=1)
fenetre.grid_rowconfigure(2, weight=1)
fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=1)
fenetre.grid_columnconfigure(2, weight=1)
```



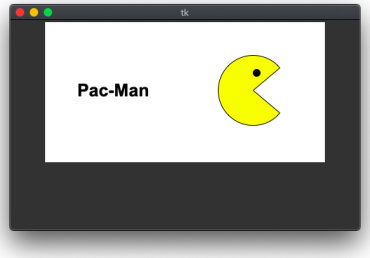
Factorisation du code et paramètre sticky

```
fenetre = tk.Tk()
fenetre.geometry('500x300')

fenetre.grid_rowconfigure(0, weight=1)
fenetre.grid_rowconfigure(1, weight=1)
fenetre.grid_rowconfigure(2, weight=1)
fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=1)
fenetre.grid_columnconfigure(2, weight=1)

etiquette_dict = {'bg':'blue', 'fg':'white', 'font':('Arial', 25, 'bold')}
grid_dict = {'sticky':'nswe', 'padx':10, 'pady':10}

etiquette1 = tk.Label(fenetre, text='Texte 1', **etiquette_dict)
etiquette2 = tk.Label(fenetre, text='Texte 2', **etiquette_dict)
etiquette3 = tk.Label(fenetre, text='Texte 3', **etiquette_dict)
etiquette4 = tk.Label(fenetre, text='Texte 4', **etiquette_dict)
etiquette5 = tk.Label(fenetre, text='Texte 5', **etiquette_dict)
etiquette6 = tk.Label(fenetre, text='Texte 6', **etiquette_dict)

etiquette1.grid(row=0, column=0, **grid_dict)
etiquette2.grid(row=0, column=1, columnspan=2, **grid_dict)
etiquette3.grid(row=1, column=0, rowspan= 2, **grid_dict)
etiquette4.grid(row=1, column=1, **grid_dict)
etiquette5.grid(row=1, column=2, **grid_dict)
etiquette6.grid(row=2, column=1, **grid_dict)

fenetre.mainloop()
```



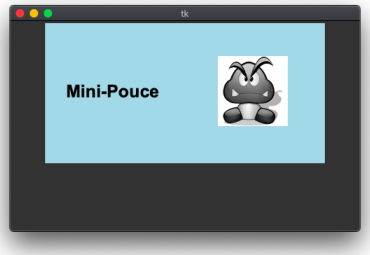
Zone graphique avec dessin et texte

```
fenetre = tk.Tk()
fenetre.geometry('500x300')

zoneGraphique = tk.Canvas(fenetre, width=400, height=200, bg='white')
zoneGraphique.create_arc(250, 50, 350, 150, outline='black', fill='yellow', start=40, extent=280)
zoneGraphique.create_oval(300, 70, 310, 80, outline='black', fill='black')
zoneGraphique.create_text(100, 100, text='Pac-Man', font='Arial 25 bold', fill='black')
zoneGraphique.pack()

fenetre.mainloop()
```



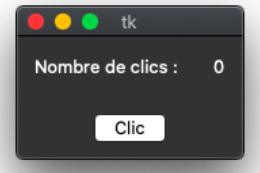
Zone graphique avec image et texte

```
fenetre = tk.Tk()
fenetre.geometry('500x300')

img = tk.PhotoImage(file='/Users/anthonyridard/Desktop/miniPouce.png')
zoneGraphique = tk.Canvas(fenetre, width=400, height=200, bg='lightblue')
zoneGraphique.create_image(300, 100, image=img)
zoneGraphique.create_text(100, 100, text='Mini-Pouce', font='Arial 25 bold', fill='black')
zoneGraphique.pack()

fenetre.mainloop()
```



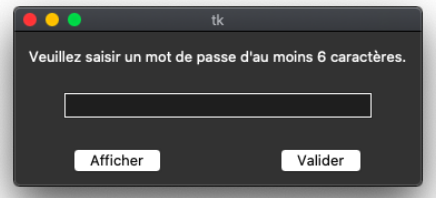
Bouton

```
etiquette1 = tk.Label(fenetre, text = 'Nombre de clics :')
etiquette2 = tk.Label(fenetre, text = '0')

bouton= tk.Button(fenetre, text = 'Clic')

grid_dict = {'padx':10, 'pady':10}

etiquette1.grid(row=0, column=0, columnspan=2, **grid_dict)
etiquette2.grid(row=0, column=2, **grid_dict)
bouton.grid(row=2, column=0, columnspan=3, **grid_dict)

fenetre.mainloop()
```



Zone de saisie et chaîne dynamique

```
fenetre = tk.Tk()

fenetre.grid_rowconfigure(0, weight=1)
fenetre.grid_rowconfigure(1, weight=1)
fenetre.grid_rowconfigure(2, weight=1)
fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=1)

etiquette = tk.Label(fenetre, text = 'Veuillez saisir un mot de passe d\'au moins 6 caractères.')

motDePasse = tk.StringVar()
motDePasse.set('')
champ = tk.Entry(fenetre, textvariable = motDePasse, show='*', width=30)

btAfficher = tk.Button(fenetre, text = 'Afficher')
btValider = tk.Button(fenetre, text = 'Valider')

grid_dict = {'padx':10, 'pady':10}

etiquette.grid(row=0, column=0, columnspan=2, **grid_dict)
champ.grid(row=1, column=0, columnspan=2, **grid_dict)
btAfficher.grid(row=2, column=0, **grid_dict)
btValider.grid(row=2, column=1, **grid_dict)

fenetre.mainloop()
```



Maintenant que nous savons construire une fenêtre avec des composants graphiques, voyons comment l'utilisateur peut interagir avec cette interface graphique.

2. Programmation évènementielle

Contrairement à une interface textuelle qui oblige l'utilisateur à saisir ses entrées une à une, une interface graphique offre à l'utilisateur plusieurs actions possibles (cliquer sur un bouton, ...). Comment le programmeur peut-il alors gérer cette situation ?

C'est la **programmation événementielle** qui apporte une solution à ce problème.

Pour illustrer ce paradigme, voici comment fonctionne la méthode **mainloop()** :

- Elle initialise l'affichage, c'est à dire parcourt la fenêtre ainsi que tous les composants qui y ont été ajoutés, et les dessine à l'écran
- Elle se met **en attente d'un événement** (clic sur un bouton, ...)
- Lorsqu'un événement se produit, elle cherche parmi tous les **composants dynamiques** celui [5] qui attend cet événement
- Le **gestionnaire d'événement** du composant concerné est alors appelé en réponse

Nous allons donc voir comment :

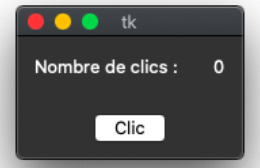
- **définir** un gestionnaire d'événement
- l'**associer** à un événement sur un composant dynamique

[5]. Il peut y en avoir plusieurs, mais simplifions ici les choses

Une bonne pratique est l'utilisation du **modèle de conception** MVC :

- **Modèle** : définition de la « logique » ne relevant pas de l'interface graphique
- **Vue** : mise en place de l'affichage (fenêtre et composants graphiques)
- **Contrôleur** : gestion des actions de l'utilisateur

Pour illustrer cette partie, reprenons l'interface suivante :



Modèle

```
compteur = 0

def suivant() :
    global compteur
    compteur = compteur + 1
    return compteur
```

Variable globale

Une variable définie dans une fonction est par défaut **locale**, c'est à dire qu'elle n'existe pas en dehors de la fonction.

Lorsque la variable est définie de manière **globale**[a], elle n'est pas détruite après l'exécution de la fonction, elle continue d'exister et est accessible à tout moment.

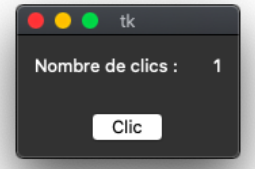
a. A éviter en général, sauf en programmation événementielle

Vue

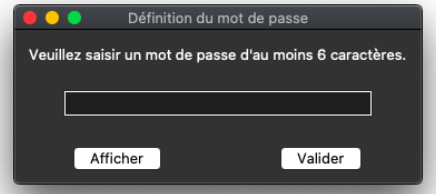
```
fenetre = tk.Tk()

fenetre.grid_rowconfigure(0, weight=1)
fenetre.grid_rowconfigure(1, weight=1)
fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=1)
fenetre.grid_columnconfigure(2, weight=1)

etiquette1 = tk.Label(fenetre, text = 'Nombre de clics :')
etiquette2 = tk.Label(fenetre, text = '0')

bouton= tk.Button(fenetre, text = 'Clic')

grid_dict = {'padx':10, 'pady':10}

etiquette1.grid(row=0, column=0, columnspan=2, **grid_dict)
etiquette2.grid(row=0, column=2, **grid_dict)
bouton.grid(row=2, column=0, columnspan=3, **grid_dict)
```

Contrôleur

```
def action_clic(e) :
    etiquette2.config(text=str(suivant()))

bouton.bind('<Button-1>', action_clic)

fenetre.mainloop()
```



Reprenons maintenant l'interface suivante :



Modèle

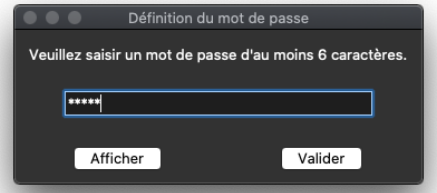
```
import tkinter as tk
import tkinter.messagebox as msg

# modèle
```

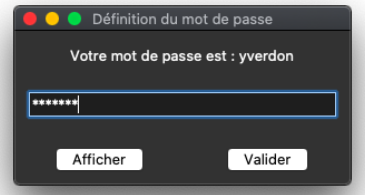
Vue

```
fenetre = tk.Tk()
fenetre.title('Définition du mot de passe')

fenetre.grid_rowconfigure(0, weight=1)
fenetre.grid_rowconfigure(1, weight=1)
fenetre.grid_rowconfigure(2, weight=1)
fenetre.grid_columnconfigure(0, weight=1)
fenetre.grid_columnconfigure(1, weight=1)

etiquette = tk.Label(fenetre, text = 'Veuillez saisir un mot de passe d\'au moins 6 caractères.')

motDePasse = tk.StringVar()
motDePasse.set('')
champ1 = tk.Entry(fenetre, textvariable = motDePasse, show='*', width=30)

btAfficher = tk.Button(fenetre, text = 'Afficher')
btValider = tk.Button(fenetre, text = 'Valider')

grid_dict = {'padx':10, 'pady':10}

etiquette.grid(row=0, column=0, columnspan=2, **grid_dict)
champ1.grid(row=1, column=0, columnspan=2, **grid_dict)
btAfficher.grid(row=2, column=0, **grid_dict)
btValider.grid(row=2, column=1, **grid_dict)
```

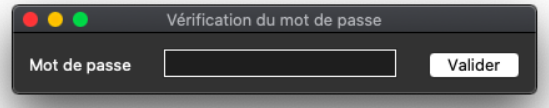
Contrôleur

```
def action_valider(e):
    if len(motDePasse.get()) < 6 :
        msg.showwarning('Message','Le mot de passe saisi est invalide')
        motDePasse.set('')
        etiquette.config(text = 'Veuillez saisir un mot de passe d\'au moins 6 caractères')
    else:
        msg.showinfo('Message', 'Votre mot de passe est bien enregistré')
        fenetre.destroy()

def action_afficher(e):
    etiquette.config(text = 'Votre mot de passe est : ' + motDePasse.get())

btAfficher.bind('<Button-1>', action_afficher)
btValider.bind('<Button-1>', action_valider)

fenetre.mainloop()
```



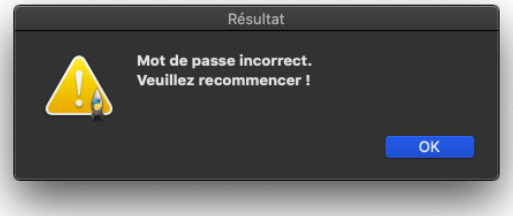


Concevoir l'interface suivante permettant de vérifier votre mot de passe[a].

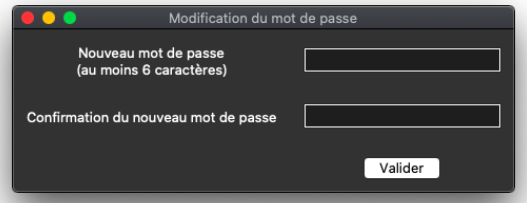


a. défini précédemment

Voici les affichages attendus :



Concevoir l'interface suivante permettant de modifier votre mot de passe.
Contrairement à l'ex précédent, vous devez créer un nouveau fichier **mvc3.py**.



Voici les affichages attendus :