PROGRAMMATION D'INTERFACES

Licence informatique & vidéoludisme

Cours préparé par:
Oumaima EL JOUBARI
Hanane ZERDOUM

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Programmation d'interfaces

01

Introduction

La programmation graphique
Le module Tkinter

02

Fenêtres

Création de fenêtres
Gestionnaire de géométrie
Techniques générales

03

Les widgets et évènements

Quelques widgets
Gestion des évènements

04

Pour aller plus loin

Animation
Audios sous Tkinter

01

Introduction

1. La programmation graphique
2. Le module Tkinter

I. Introduction

1. La programmation graphique

Utilité d'une GUI : Une application graphique ou GUI (*graphical user interface*) est une interface permettant d'interagir avec un programme sans avoir à saisir des lignes de commandes.

La fenêtre : Un élément de l'interface graphique d'un programme. Elle est composée de deux parties: zone cliente et zone non cliente.

Le widget: Un widget (*window gadget*) est un objet graphique inclus dans la fenêtre permettant à l'utilisateur d'interagir avec votre programme de manière conviviale.

➔ Exemples: Boutons, listes de choix, zone de texte….

I. Introduction

1. La programmation graphique

La programmation événementielle

Dans une application graphique, l'exécution est décidée par l'utilisateur en fonction de ses interactions avec les différents widgets. Le programme attend donc que l'utilisateur déclenche une action. On appelle cette action un événement.

➔ Exemples d'évènements:

- Un clic sur un bouton de la souris
- Le déplacement de la souris
- L'appui sur une touche du clavier
- Un clic sur la croix de fermeture de la fenêtre principale

I. Introduction

1. La programmation graphique

La programmation événementielle

Dans une application graphique, l'exécution est décidée par l'utilisateur en fonction de ses interactions avec les différents widgets. Le programme attend donc que l'utilisateur déclenche une action. On appelle cette action un événement.

Le gestionnaire d'événements : C'est une sorte de « boucle infinie » qui est à l'attente d'événements provoqués par l'utilisateur. C'est lui qui effectuera une action lors de l'interaction de l'utilisateur avec chaque widget de la GUI. Ainsi, l'exécution du programme sera réellement guidée par les actions de l'utilisateur.

I. Introduction

1. La programmation graphique

Modules pour construire des applications graphiques en Python:

- ➔ Tkinter
- ➔ wxpython
- ➔ PyQt
- ➔ PyGObject
- ➔ …

I. Introduction

2. Le module Tkinter

- Tkinter (*tool kit interface*) est une bibliothèque écrite en Python permettant la création d'interfaces graphiques.
- Tkinter est présent de base dans les distributions Python, donc pas besoin a priori de faire d'installation de module externe.

Exemple

```python
#Importer le module tkinter
import tkinter as tk

#Creer l'application en tant que classe
class Application(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self.creer_widgets()
    def creer_widgets(self):
        self.label = tk.Label(self, text="J'adore Python !")
        self.bouton = tk.Button(self, text="Quitter", command=self.quit)
        self.label.pack()
        self.bouton.pack()

app = Application()
app.title("Ma Première App")
app.mainloop()
```

fichier GUI1.py

02

Fenêtres

1. Création de fenêtres
2. Gestionnaire de géométrie
3. Techniques générales

II. Fenêtres

1. Création de fenêtres

➔ **Créer une application graphique**

- ❖ Importer le module Tkinter;
- ❖ Créer une classe Application qui hérite de la classe Tk;
- ❖ Créer le constructeur de la classe Application en appelant le constructeur de la classe mère;
- ❖ Instancier la classe Application.

Tk est une classe définie dans le module tkinter et qui permet la création de la fenêtre-maîtresse de l'application.

```
#Importer le module tkinter
import tkinter as tk

#Creer l'application en tant que classe
class Application(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

app = Application()
```

fichier GUI2.py

II. Fenêtres

1. Création de fenêtres

➔ **Activer une fenêtre**

- ❖ Pour que la fenêtre continue à apparaître, il faut utiliser la méthode *mainloop()*.
- ❖ Cette méthode va lancer le gestionnaire d'événements qui interceptera la moindre action de l'utilisateur.
- ❖ Elle est souvent à la fin du script, puisqu'on écrit d'abord le code construisant l'interface, et on lance le gestionnaire d'événements pour lancer l'application.

```
#Creer l'application en tant que classe
class Application(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

app = Application()
app.mainloop()
```

fichier GUI2.py

II. Fenêtres

1. Création de fenêtres

➔ Modifier le titre de la fenêtre

Par défaut, le bandeau d'une fenêtre porte le titre de tk;

- ❖ Pour modifier le titre de la fenêtre, on utilise la méthode *title()*.

```
#Creer l'application en tant que classe
class Application(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

app = Application()
app.title("Ma première App!")
```

fichier GUI3.py

II. Fenêtres

1. Création de fenêtres

➔ Définir la taille de la fenêtre

Pour définir la taille de la fenêtre, on utilise le widget Canvas;

- ❖ Importer la classe *Canvas*;
- ❖ Instancier la classe *Canvas* en précisant la taille de la fenêtre;
- ❖ Placer le widget Canvas dans la fenêtre avec la méthode *pack()*.

```python
#Importer la classe Canvas
from tkinter import Canvas

#Creer l'application en tant que classe
class Application(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

app = Application()
cnv = Canvas(app, width = 300, height = 300)
cnv.pack()
```

fichier GUI4.py

II. Fenêtres

1. Création de fenêtres

➔ Redimensionnement de la fenêtre

Une fenêtre Tk est par défaut redimensionnable.

- ❖ Pour bloquer le redimensionnement on utilise la méthode *resizable()*.
- ❖ Le premier paramètre est pour restreindre le changement de la hauteur, et le 2ème pour la largeur.
- ❖ On peut passer en argument “0” ou “False” pour bloquer le redimensionnement.

```
app = Application()
cnv = Canvas(app, width = 300, height = 300)
cnv.pack()

app.resizable(False, False)
```

fichier GUI5.py

II. Fenêtres

1. Création de fenêtres

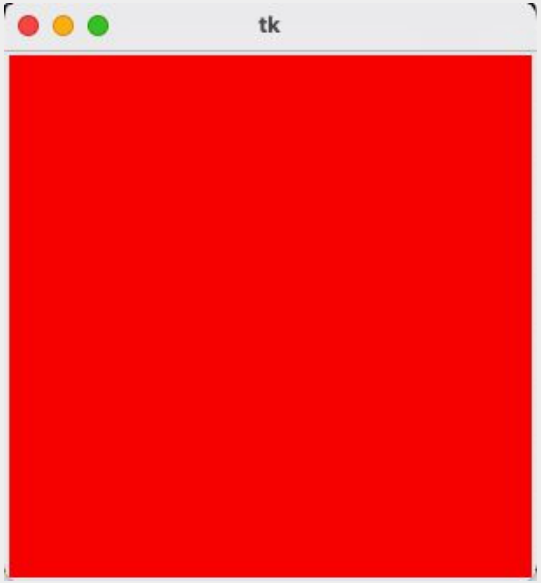
➔ Définir la couleur du background

Pour donner une couleur au background d'une fenêtre, on utilise le widget *Canvas*:

- ❖ On peut spécifier la couleur avec des noms comme "red , blue,...", et on peut aussi la spécifier comme des codes (#49A ou #0059b3).

```
app = Application()
cnv = Canvas(app, width = 300, height = 300, bg = "red")
cnv.pack()
```

fichier GUI6.py

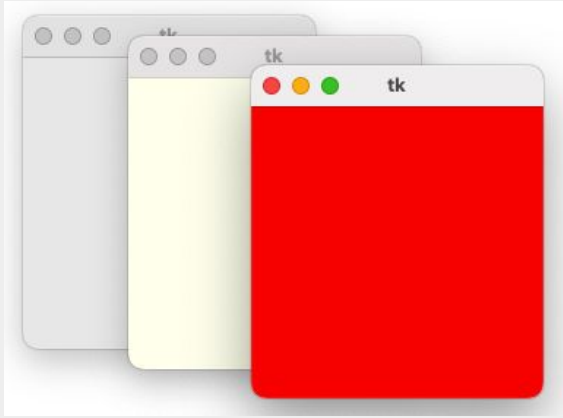


II. Fenêtres

1. Création de fenêtres

➔ Ouvrir plusieurs fenêtres

Pour ouvrir plusieurs fenêtres au lancement de votre application, il faut utiliser le widget *TopLevel* au lieu de faire plusieurs appels successifs au constructeur *Tk*.



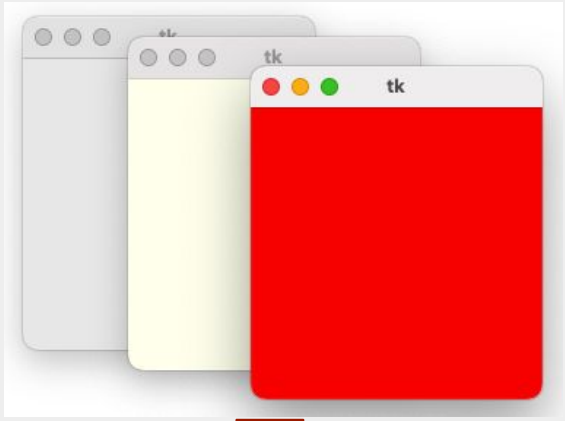

```
app = Application()
a = Toplevel(app, bg="red")
b = Toplevel(app, bg="ivory")
```

fichier GUI7.py

II. Fenêtres

1. Création de fenêtres

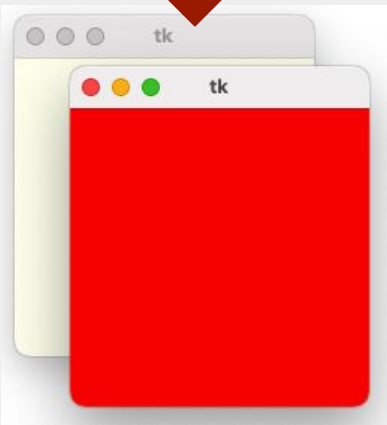
➔ Ouvrir plusieurs fenêtres



On remarque que la fenêtre principale s'affiche avec deux autres fenêtres. Pour cacher la fenêtre maîtresse, utiliser la méthode *withdraw()*:

```
app = Application()
a = Toplevel(app, bg="red")
b = Toplevel(app, bg="ivory")
app.withdraw()
```

fichier GUI7.py



II. Fenêtres

1. Création de fenêtres

➔ **Ouvrir plusieurs fenêtres**

- Comme la fenêtre est cachée, il n'y a plus moyen de fermer définitivement l'application.
- On peut y remédier de la manière suivante :

```
class Application(tk.Tk):

    closed=[False,False]

    def __init__(self):
        tk.Tk.__init__(self)
    @classmethod
    def quit_a(cls):
        a.destroy()
        if cls.closed[1]:
            app.destroy()
        else:
            cls.closed[0] = True
    @classmethod
    def quit_b(cls):
        b.destroy()
        if cls.closed[0]:
            app.destroy()
        else:
            cls.closed[1] = True

app = Application()
a = Toplevel(app, bg="red")
b = Toplevel(app, bg="ivory")

a.protocol("WM_DELETE_WINDOW",Application.quit_a)
b.protocol("WM_DELETE_WINDOW",Application.quit_b)

app.withdraw()
app.mainloop()
```

fichier GUI8.py

II. Fenêtres

1. Création de fenêtres

➔ **Le mode plein écran (Fullscreen)**

- On peut placer une fenêtre en mode plein écran avec l'option "*fullscreen*".
- La sortie du mode plein écran n'est pas prévue par défaut donc il faut l'écrire soit même.
- Dans cet exemple, on a lié la touche Echap au retour de l'écran à sa position normale.

```
class Application(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)

    def normalscreen():
        app.attributes("-fullscreen", False)

app = Application()
app.attributes("-fullscreen", True)

app.bind("<Escape>", Application.normalscreen)
app.mainloop()
```

fichier GUI9.py

II. Fenêtres

1. Création de fenêtres

➔ **Supprimer une fenêtre**

- On a deux façons différentes pour supprimer une fenêtre:
  - La méthode *destroy()*: permet de détruire la fenêtre.
  - La méthode *quit()*: détruit non seulement la fenêtre mais aussi tous les objets placés sur la fenêtre.

```
class Application(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        self.creerwidget()

    def creerwidget(self):
        cnv = Canvas(self, width=200, height=200, bg="ivory")
        cnv.pack()
        Button(self, text="Quitter", command=self.destroy).pack()

app = Application()

app.mainloop()
```

fichier GUI10.py

II. Fenêtres

2. Gestionnaire de géométrie

- ➔ Pour contrôler la dimension et placer ses widgets à l'interieur de la fenetre, il existe trois gestionnaires de géométrie :
  - ◆ pack (empilement vertical ou horizontal des widgets)
  - ◆ grid (dispose les widgets selon une grille)
  - ◆ place (dispose les widgets à une position définie)
- ➔ Tant qu'un widget n'est pas associé à un gestionnaire de géométrie, il n'apparaît pas à l'écran.
- ➔ Les gestionnaires de géométrie sont incompatibles entre eux, on ne peut utiliser qu'un seul type de gestionnaire dans une fenêtre ou cadre.

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **pack**:

- Ce gestionnaire regroupe tous les widgets les uns après les autres.
- On peut utiliser trois options pour contrôler ce gestionnaire de géométrie:

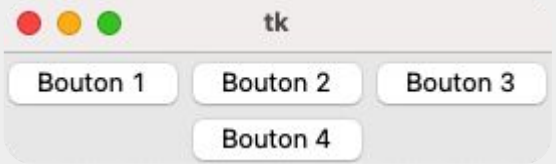
expand	**fill**	**side**
Défini sur *True* pour que le widget se développe et remplit tout espace non utilisé dans le widget parent.	Détermine si le widget remplit tout espace supplémentaire ou garde ses propres dimensions minimales: *NONE* (par défaut), *X* (remplir horizontalement), *Y* (remplir verticalement), *BOTH* (remplir horizontalement et verticalement)	Détermine le côté du widget parent: *TOP* (par défaut) *BOTTOM* *LEFT* *RIGHT*

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **pack**:

```
class Application(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        self.creerwidget()

    def creerwidget(self):
        frame = Frame(self)
        frame.pack(side = BOTTOM)

        btn1 = Button(self, text="Bouton 1")
        btn1.pack(side = LEFT)

        btn2 = Button(self, text="Bouton 2")
        btn2.pack(side = LEFT)

        btn3 = Button(self, text="Bouton 3")
        btn3.pack(side = LEFT)

        btn4 = Button(frame, text="Bouton 4")
        btn4.pack(side = BOTTOM)

app = Application()
app.mainloop()
```



fichier GUI11.py

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **grid**:

- Ce gestionnaire divise le widget maître en un certain nombre de lignes et de colonnes, et chaque cellule de la grille peut contenir un widget.

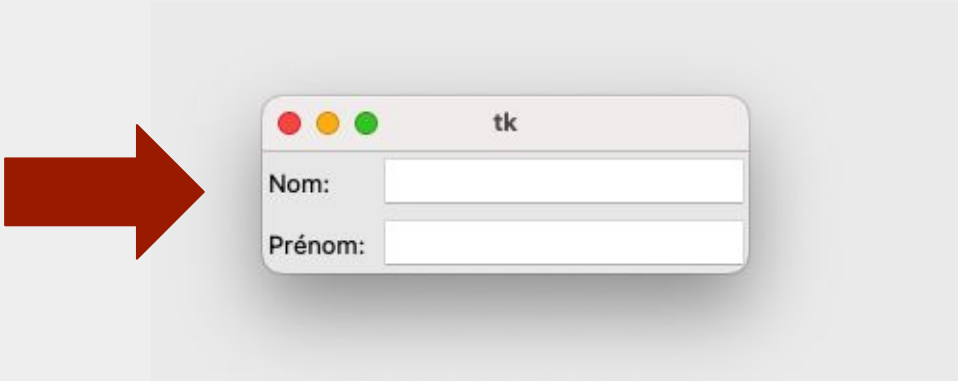
column	row	padx/pady	ipadx/ipady	rowpan/columnspan	Sticky
La colonne dans laquelle placer le widget; 0 par défaut (colonne à gauche)	La ligne dans laquelle placer le widget; par défaut la première ligne qui est encore vide.	Combien de pixels pour remplir le widget, horizontalement et verticalement, à l'extérieur des bordures du widget.	Combien de pixels pour remplir le widget, horizontalement et verticalement, à l'intérieur des bordures du widget.	Le nombre de lignes/colonnes le widget occupe (1 par défaut)	Étirement du widget dans le cas où la cellule est plus large que sa taille: N, E, S, W, NE, NW, SE, et SW

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **grid**:

```
class Application(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        self.creerwidget()

    def creerwidget(self):
        l1 = Label(self, text = "Nom: ")
        l2 = Label(self, text = "Prénom: ")
        e1 = Entry(self)
        e2 = Entry(self)

        l1.grid(row = 0, column = 0, sticky = W, pady=2)
        l2.grid(row = 1, column = 0, sticky = W, pady=2)

        e1.grid(row = 0, column = 1, sticky = W, pady=2)
        e2.grid(row = 1, column = 1, sticky = W, pady=2)


app = Application()
app.mainloop()
```



fichier GUI12.py

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **place**:

- Ce gestionnaire organise les widgets en les plaçant dans une position spécifique dans le widget maître.

anchor	bordermode	height/width	relheight/relwidth	relx/rely	x / y
L'emplacement exact du widget : peut être N, E, S, W, NE, NW, SE, SW ou CENTER, la valeur par défaut est NW.	INSIDE (par défaut) pour indiquer que d'autres options font référence à l'intérieur du parent (en ignorant la bordure du parent); OUTSIDE autrement.	Hauteur et largeur en pixels.	Hauteur et largeur sous forme d'un nombre flottant entre 0,0 et 1,0	Décalage horizontal et vertical sous forme d'un nombre flottant entre 0,0 et 1,0	Décalage horizontal et vertical en pixels.

II. Fenêtres

2. Gestionnaire de géométrie

➔ Le gestionnaire **place**:

```
def creerwidget(self):
    frame = Frame(self, width=200, height=200, bg="ivory")
    frame.place(anchor = NW)

    l1 = Label(frame, text = "Bonjour", bg="ivory")
    l2 = Label(frame, text = "Bonjour", bg="ivory")
    l3 = Label(frame, text = "Bonjour", bg="ivory")

    l1.place(x = 10, y = 10)
    l2.place(x = 50, y = 50)
    l3.place(x = 90, y = 90)
```

fichier GUI13.py

II. Fenêtres

3. Techniques générales

➔ Les couleurs:

Il existe deux codages des couleurs sous Tkinter :

- **Nom de couleur :** les couleurs standard du html, peuvent être appelées par leur nom, typiquement des noms courants “*red*” ou d’autres comme “*ivory*”;
- **Codage hexadécimal :** on fournit une chaîne hexadécimale RGB commençant par le caractère # ; il existe plusieurs formes, la plus simple étant un code à 3 chiffres hexadécimaux, du type "#5fc" où chacune des trois composantes R, G et B est représentée par un seul chiffre hexadécimal.

II. Fenêtres

3. Techniques générales:

➔ Les couleurs:

fichier Colors.py

snow	deep sky blue	gold	seashell3	SlateBlue2	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70
antique white	pale turquoise	sandy brown	bisque3	blue1	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71
papaya whip	dark turquoise	dark salmon	bisque4	blue2	PaleTurquoise4	chartreuse4	IndianRed1	orange2	PaleVioletRed3	thistle1	gray34	gray72
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray36	gray74
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab4	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75
navajo white	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3		gray38	gray76
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4		gray39	gray77
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1		gray40	gray78
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2		gray42	gray79
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3		gray43	gray80
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4		gray44	gray81
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2		gray45	gray82
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3		gray46	gray83
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4		gray47	gray84
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1		gray48	gray85
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2		gray49	gray86
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3		gray50	gray87
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4		gray51	gray88
light grey	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1		gray52	gray89
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2		gray53	gray90
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3		gray54	gray91
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4		gray55	gray92
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1		gray56	gray93
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2		gray57	gray94
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3		gray58	gray95
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4		gray59	gray97
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1		gray60	gray98
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2		gray61	gray99
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3		gray62	
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4		gray63	

II. Fenêtres

3. Techniques générales:

- ➔ Les polices:
  - L'option *font* des widgets permet de définir la fonte du texte.
  - Pour décrire une fonte, il faut donner son nom (ex: Arial, Comic Sans Ms…), sa taille et ses attributs (Bold, Italic,...).
  - Elle admet trois syntaxes :
    - font = "Times 12 bold"
    - font = "{Times} 12 bold" à utiliser si le nom de la police contient des espaces
    - font = ("Times", 12, "bold") à utiliser si le nom de la police contient des espaces

La taille est un entier positif si elle est exprimée en point. Une taille négative exprime une taille en pixels.

II. Fenêtres

3. Techniques générales:

➔ Les polices:

```
class Application(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        self.creerwidget()

    def creerwidget(self):
        Label(self, text="ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      font="Arial 12").pack(side=TOP, anchor="w")
        Label(self, text="ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      font=("Times New roman", 12, "bold")).pack(side=TOP, anchor="w")
        Label(self, text="ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      font=("{Times New roman} 12 italic")).pack(side=TOP, anchor="w")
        Label(self, text="ABCDEFGHIJKLMNOPQRSTUVWXYZ",
      font=("Courier")).pack(side=TOP, anchor="w")
```

fichier GUI14.py