PROGRAMMATION D'INTERFACES

Licence informatique & vidéoludisme

Cours préparé par:
Oumaima EL JOUBARI
Hanane ZERDOUM

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Programmation orientée objet

01

Introduction

Introduction à la POO
Concepts de base de la POO

02

Classes et instances

Définition des classes
Constructeur de classe

03

Les méthodes

Définition et appel des méthodes

04

Héritage et polymorphisme

Héritage
Classes et sous-classes
Polymorphisme et surcharge

01

Introduction

1. Introduction à la POO
2. Concepts de base de la POO

I. Introduction

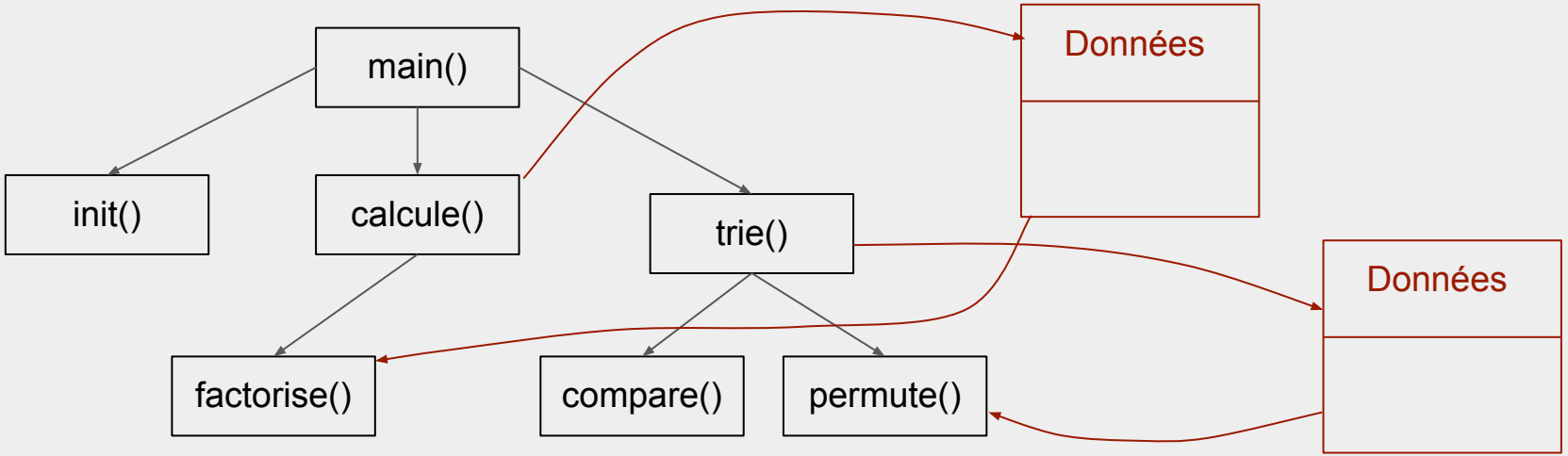
1. Introduction à la POO

La programmation procédurale (PP)

La PP repose sur l'équation suivante:

Programme = Structures de données + Algorithmes

➔ Consiste à décomposer le programme en fonctions (modules) simples.



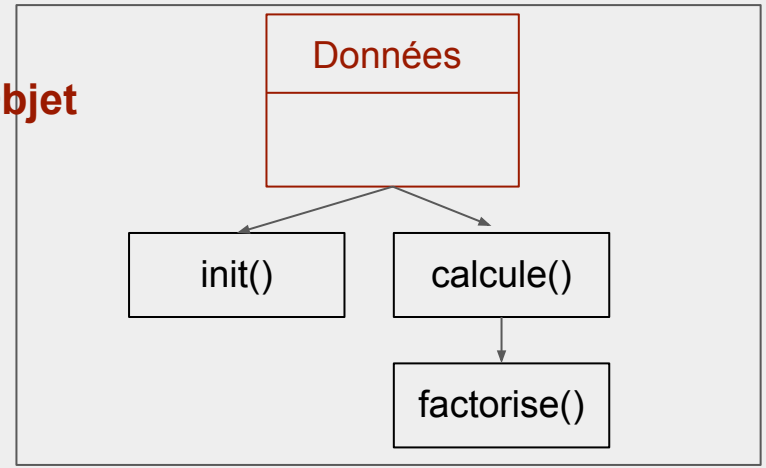
I. Introduction

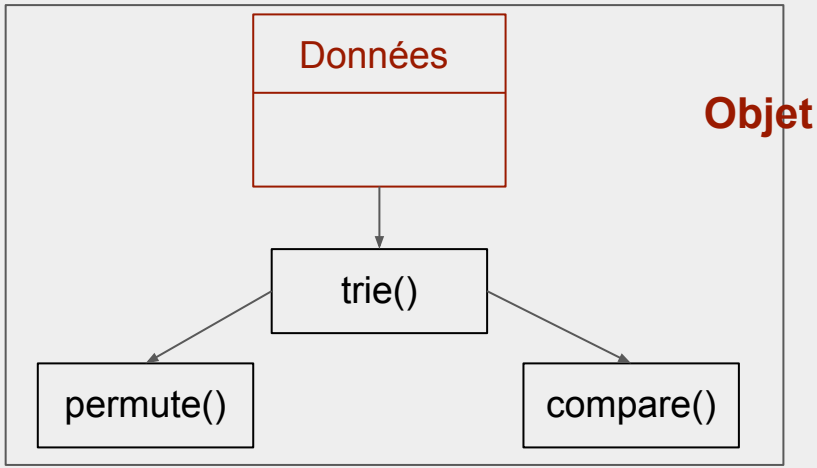
1. Introduction à la POO

La programmation orientée objet (POO)

La POO est basée sur les données:

➔ Le programme détermine les données à traiter et les fonctions qui permettent de les manipuler.





I. Introduction

1. Introduction à la POO

La programmation orientée objet (POO)

Objet= Données + Méthodes

➔ Un objet est une association de données et de fonctions (méthodes) qui agissent sur ces données.

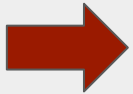
➔ La POO est donc une programmation dans laquelle un programme est organisé comme un ensemble d'objets coopérant ensemble.

I. Introduction

2. Concepts de base de la POO

➔ Un objet a:

- ◆ Ses propres données (*attributs*) qui peuvent être des données simples (entier, chaîne de caractère, …) ou d'autres objets.
- ◆ Ses propres fonctions membres (*méthodes*) qui représente son comportement.
  - Ce sont les traitements qu'on peut appliquer aux données.
- ◆ Une *identité* qui permet de l'identifier parmi les autres objets.



Voiture1

Marque: Lamborghini
Modèle: Aventador
Couleur: Noir
Etat: En Marche

Démarrer()
Arrêter()
Peindre(NouvColor)
GetEtat()
…

I. Introduction

2. Concepts de base de la POO

- ➔ Objets prédéfinis par Python:
  - ◆ Entiers, listes, booléens, chaînes de caractères...
- ➔ Pour créer un nouveau type d'objets, il faut définir à quoi il ressemble = *définir une classe*.
- ➔ Les *classes* servent de « moules » pour la création des objets.
- ➔ Les objets qui ont les mêmes états et les mêmes comportements sont regroupés : classe.

02

Classes et instances

1. Définition d'une classe
2. Constructeur de classe

II. Classes et instances

1. Définition d'une classe:

Classe

LIVRE

Titre (chaîne de caractères)
Date de parution (entier)
Auteur (chaîne de caractères)

Instances

livre1

Titre : "Le pouvoir du moment présent"
Date de parution : 2010
Auteur : "Eckhart Tolle"

livre2

Titre : "Père riche père pauvre"
Date de parution : 2017
Auteur : "Robert T. Kiyosaki"

livre3

Titre : "L'alchimiste"
Date de parution : 2001
Auteur : "Paulo Coelho"

II. Classes et instances

1. Définition d'une classe:

➔ Créer une classe:

```python
#Définition de la classe Voiture
class Voiture:
    #Les attributs de la classe
    marque = "Lamborghini"
    modele = "Aventador"
    couleur = "Noir"

#Création de deux instances de la classe Voiture
voiture_1 = Voiture()
voiture_2 = Voiture()

#Afficher la marque des deux voitures crées
print(voiture_1.marque)
print(voiture_2.marque)
```

Exécution

```
(base) Oumaimas-MacBook-Air:desktop oumaima$ python Voiture.py
Lamborghini
Lamborghini
```

fichier Voiture1.py

II. Classes et instances

2. Constructeur de la classe:

➔ Créer un constructeur de classe:

```python
#Définition de la classe Voiture
class Voiture:
    #création du constructeur
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

#Création de deux instances de la classe Voiture
voiture_1 = Voiture("Lamborghini" , "Noir")
voiture_2 = Voiture("BMW", "Rouge")

#Afficher la marque des deux voitures crées
print(voiture_1.marque)
print(voiture_1.couleur)
print(voiture_2.marque)
print(voiture_2.couleur)
```

Exécution

```
(base) Oumaimas-MacBook-Air:partie2 oumaima$ python Voiture2.py
Lamborghini
Noir
BMW
Rouge
```

fichier Voiture2.py

03

Les méthodes

Définition et appel des méthodes

III. Les méthodes

- Définition et appel des méthodes

```python
#Definition de la classe Voiture
class Voiture:

    #creation du constructeur
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

    #Definition des methodes

    def peindre(self, NouvCouleur):
        self.couleur = NouvCouleur

#Creation de deux instances de la classe Voiture
voiture_1 = Voiture("Lamborghini" , "Noir")

#Afficher la marque des deux voitures crees
print(voiture_1.couleur)
voiture_1.peindre("Rouge")
print(voiture_1.couleur)
```

Exécution

```
(base) Oumaimas-MacBook-Air:partie2 oumaima$ python Voiture4.py
Noir
Rouge
```

fichier Voiture3.py

04

Héritage et polymorphisme

1. Héritage
2. Classes et sous-classes
3. Polymorphisme et surcharge

IV. Polymorphisme et héritage

1. Définition et intérêts

➔ Définition:

Technique offerte par les langages de programmation pour construire une classe à partir d'une autre en partageant ses attributs et méthodes.

➔ Intérêts:

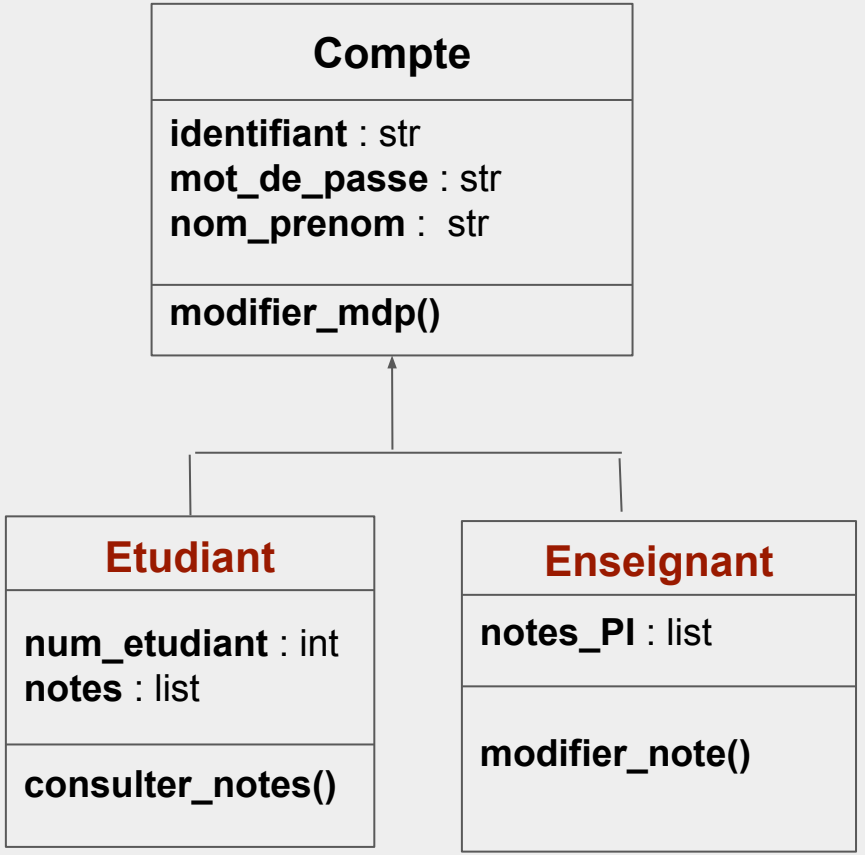
- **Spécialisation:** une nouvelle classe hérite les attributs/méthodes de la classe mère mais on peut lui ajouter de nouveaux attributs/méthodes.
- **Redéfinition:** une nouvelle classe peut redéfinir les attributs/méthodes d'une classe de manière à en changer le sens ou le comportement pour le cas particulier défini par la nouvelle classe.
- **Réutilisation:** évite d'avoir à répéter du code existant de façon inutile.

IV. Polymorphisme et héritage

2. Classes et sous-classes

➔ <u>Définitions:</u>

- Les classes *Etudiant* et *Enseignant* héritent de la classe *Compte*.
- La classe *Compte* est appelée la classe mère.
- Les classes *Etudiant* et *Enseignant* sont appelées les classes filles.
- La classe Compte est la super-classe des classes *Etudiant* et *Enseignant*.
- *Etudiant* et *Enseignant* sont les sous-classes de la classe *Compte.*

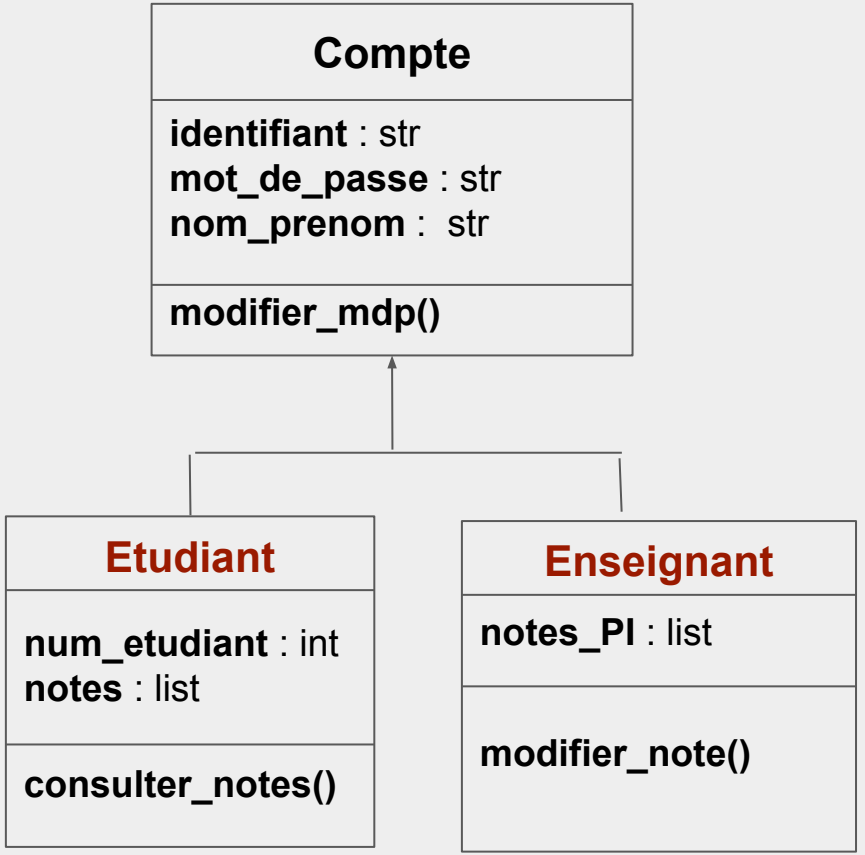


IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Définition de la classe Compte
class Compte:
    def __init__(self, id, mdp, nom_prenom):
        self.id = id
        self.mdp = mdp
        self.nom_prenom = nom_prenom

    def modifier_mdp(self, NouvMdp):
        self.mdp = NouvMdp
```

fichier Compte.py

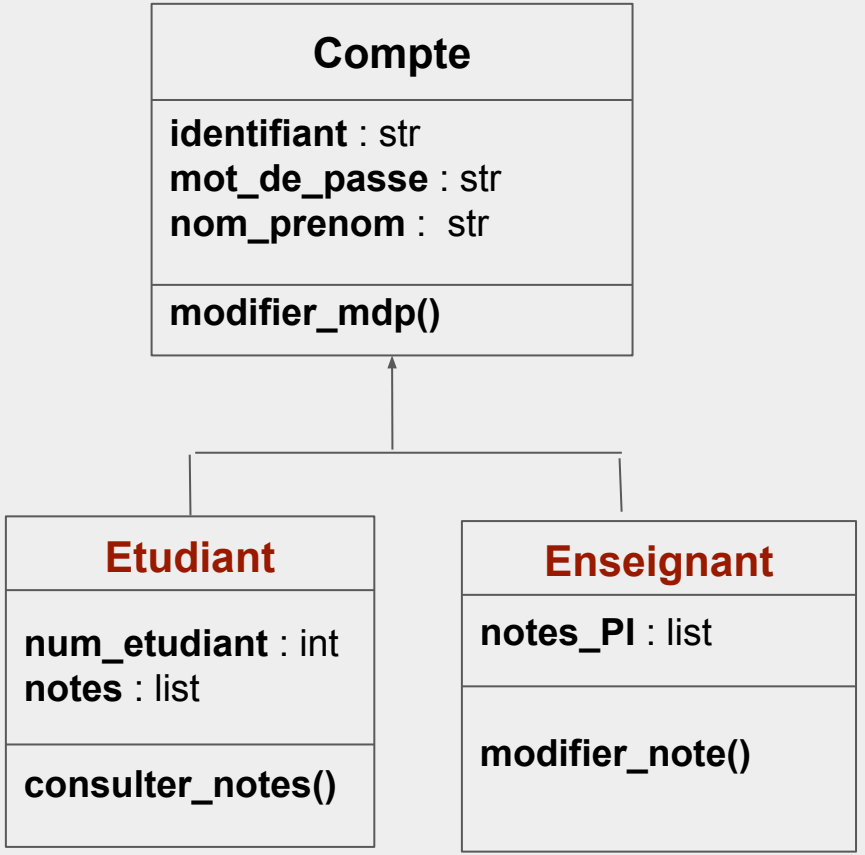


IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Definition de la classe Etudiant
class Etudiant(Compte):
    def __init__(self, id, mdp, nom_prenom, num_etudiant, notes):
        super().__init__(id, mdp, nom_prenom)
        self.num_etudiant = num_etudiant
        self.notes = notes

    def consulter_notes(self):
        print(self.notes)
```

fichier Compte.py

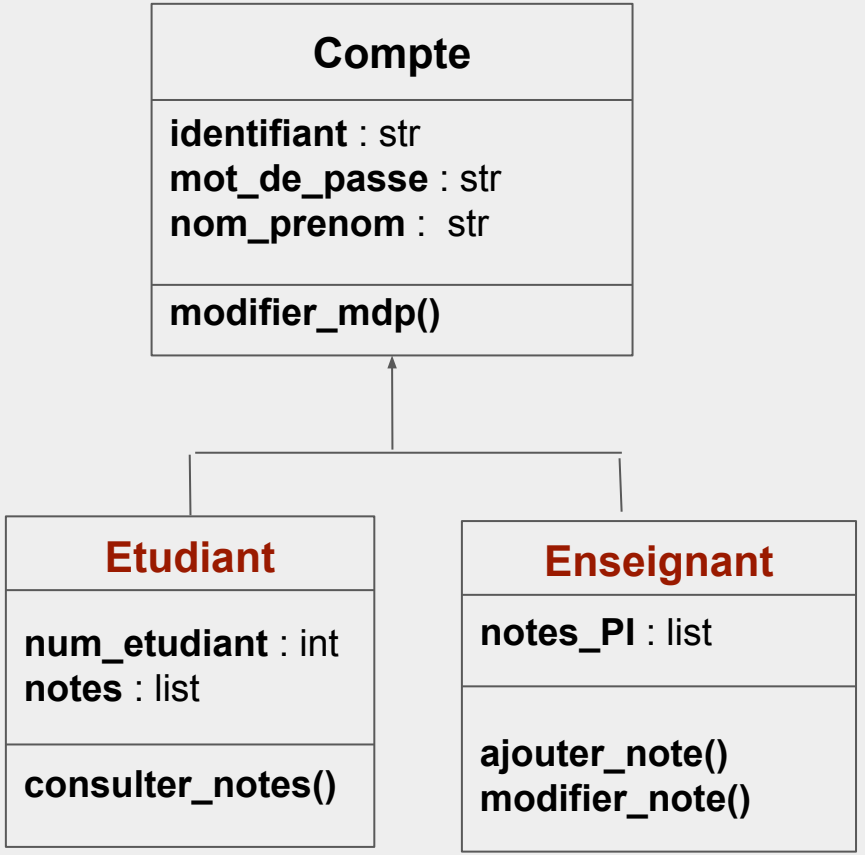


IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Definition de la classe Enseignant
class Enseignant(Compte):
    def __init__(self, id, mdp, nom_prenom, notes_PI):
        super().__init__(id, mdp, nom_prenom)
        self.notes_PI = []

    def ajouter_note(self, note):
        self.notes.append(note)

    def modifier_note(self, NouvNote, etudiant):
        self.notes_PI[etudiant] = NouvNote
```

fichier Compte.py



IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

➔ Définition du polymorphisme: C'est la capacité d'une méthode à se comporter différemment en fonction de l'objet qui lui est passé.

➔ Exemple: La méthode *sorted()*

Cette méthode trie par ordre ASCII les chaînes de caractères et par ordre croissant les listes d'entiers:

```
liste_triee = sorted([13,4,78,12,98])
```

```
liste_triee
```

```
[4, 12, 13, 78, 98]
```

```
liste_triee = sorted("bonjour")
```

```
liste_triee
```

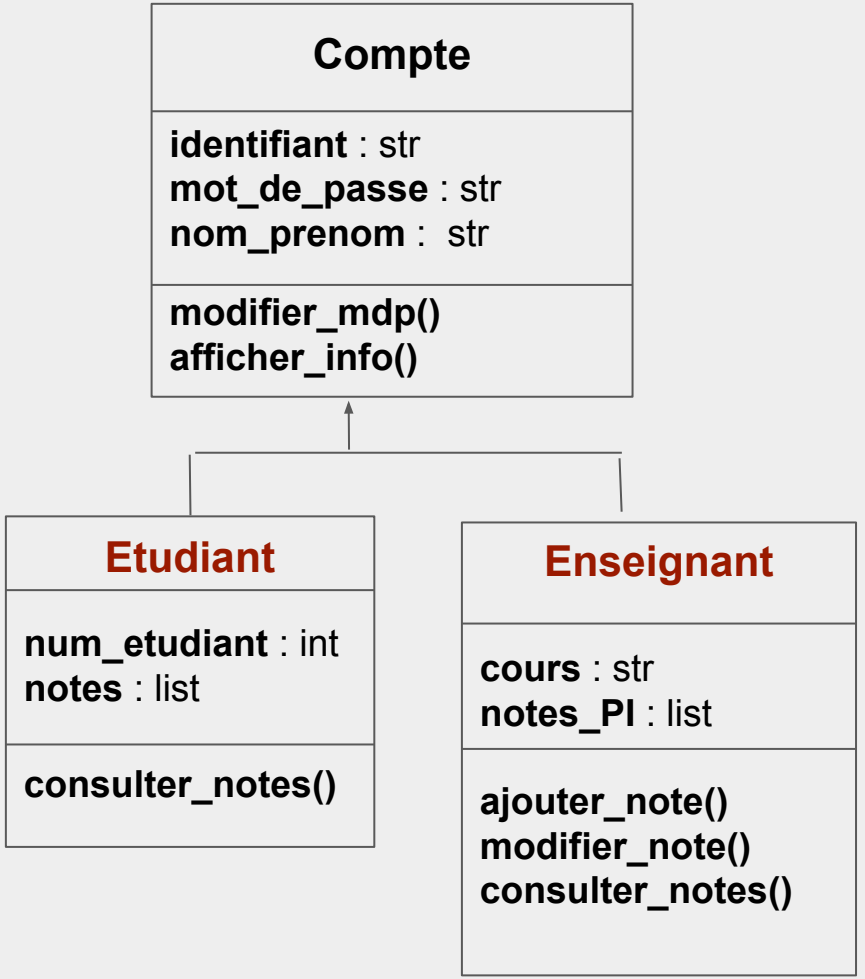
```
['b', 'j', 'n', 'o', 'o', 'r', 'u']
```

IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

Définition de la surcharge: possibilité de définir des méthodes possédant le même nom mais des arguments différents.

Redéfinition (overriding): lorsque la sous-classe définit une méthode dont le nom et les paramètres sont identiques.

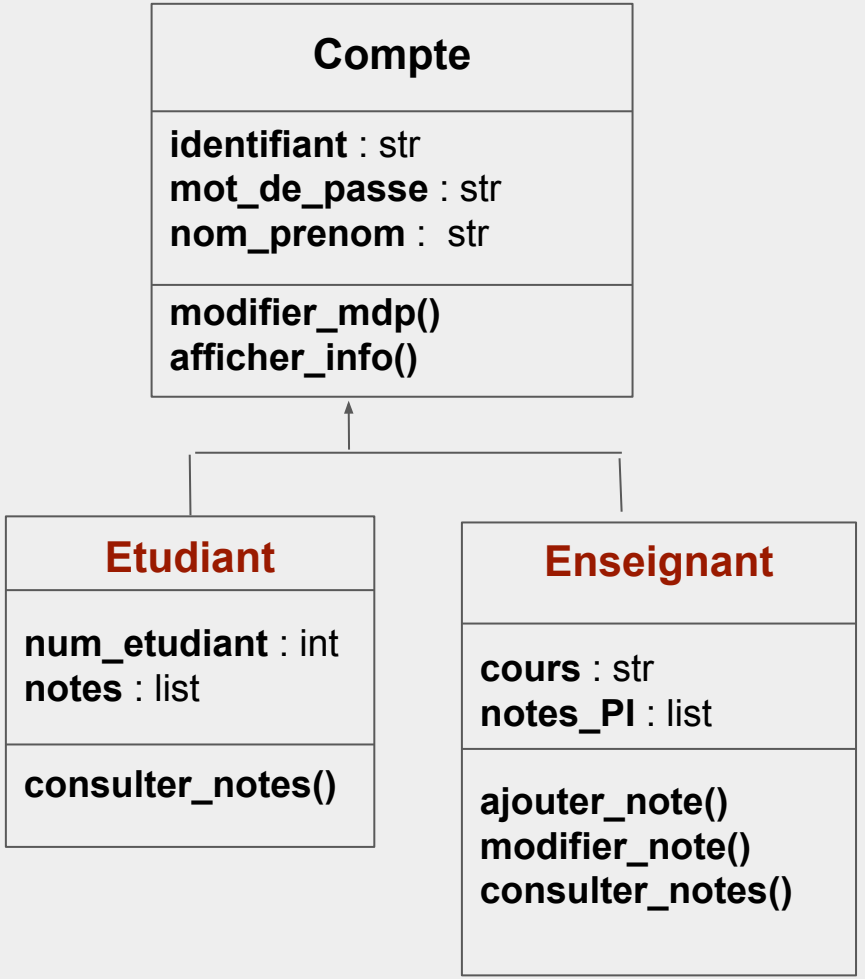


IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
class Compte:
    def __init__(self, id, mdp, nom_prenom):
        self.id = id
        self.mdp = mdp
        self.nom_prenom = nom_prenom

    def modifier_mdp(self, NouvMdp):
        self.mdp = NouvMdp

    def afficher_info(self):
        print("Identifiant: ", self.id)
        print("Nom et prenom: ", self.nom_prenom)
```

fichier Compte.py

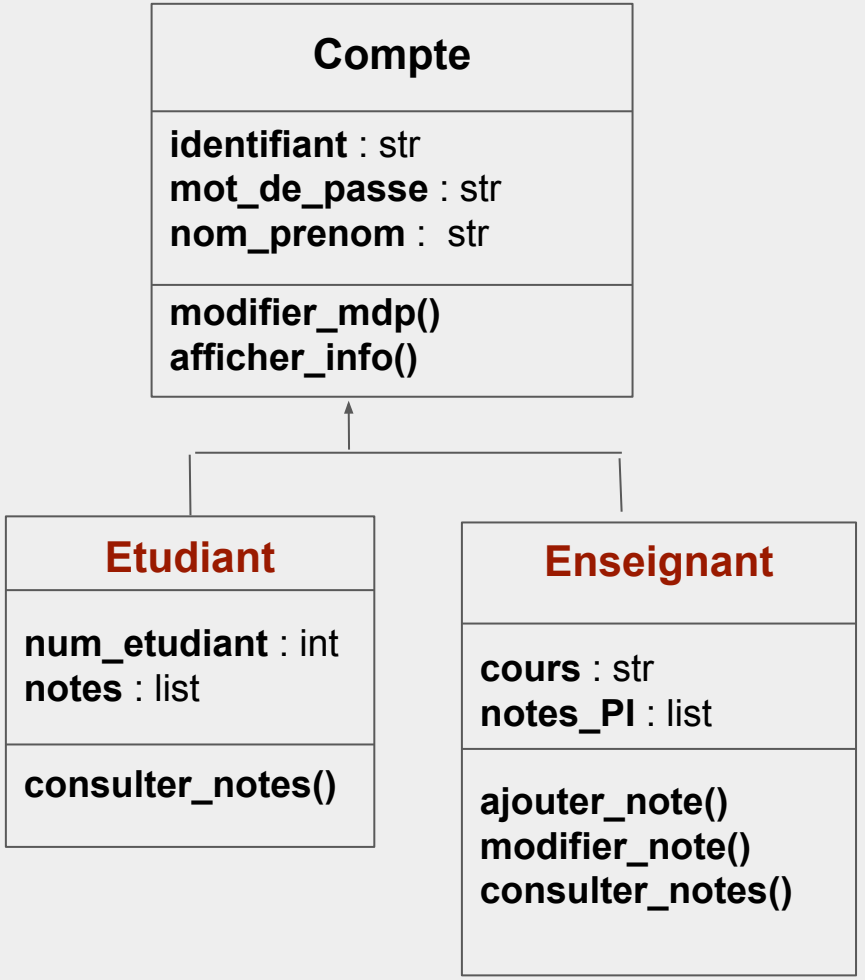


IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
#Definition de la classe Etudiant
class Etudiant(Compte):
    def __init__(self, id, mdp, nom_prenom, num_etudiant, notes):
        super().__init__(id, mdp, nom_prenom)
        self.num_etudiant = num_etudiant
        self.notes = notes

    def consulter_notes(self):
        print(self.notes)

    def afficher_info(self):
        super().afficher_info()
        print("Numero-etudiant: ", self.num_etudiant)
```

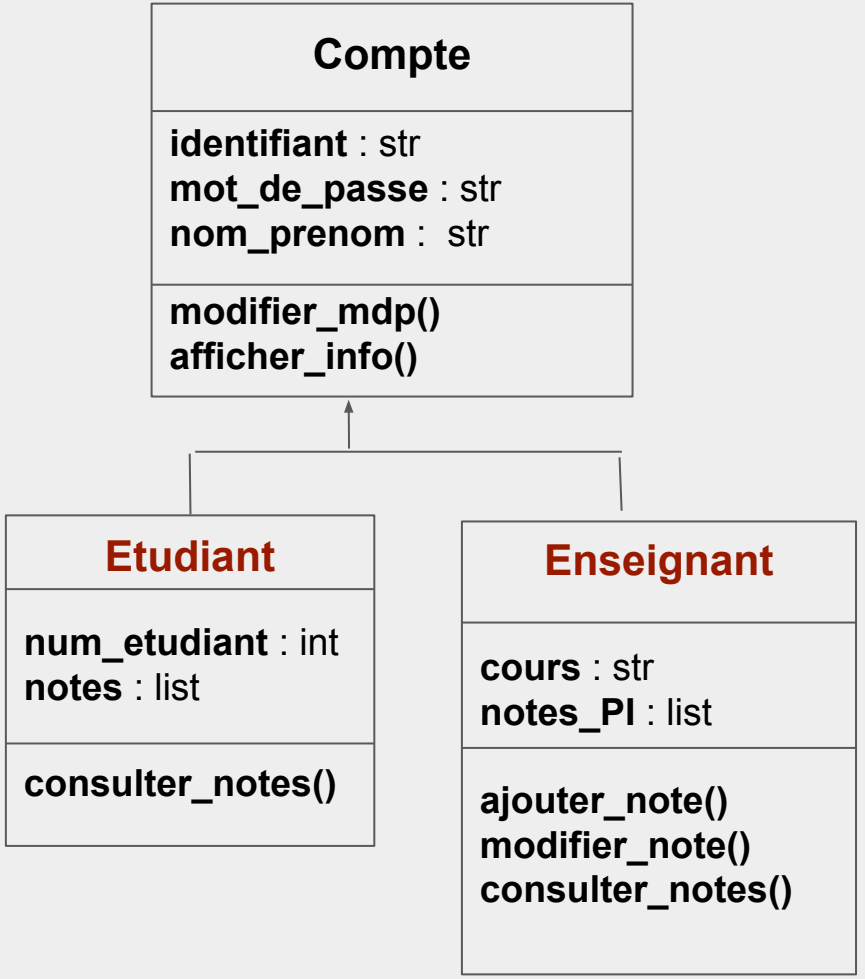
fichier Compte.py



IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
#Definition de la classe Enseignant
class Enseignant(Compte):
    def __init__(self, id, mdp, nom_prenom, cours, notes_PI):
        super().__init__(id, mdp, nom_prenom)
        self.cours = cours
        self.notes_PI = []

    def ajouter_note(self, note):
        self.notes.append(note)

    def modifier_note(self, NouvNote, etudiant):
        self.notes_PI[etudiant] = NouvNote

    def consulter_notes(self):
        print(self.notes_PI)

    def afficher_info(self):
        super().afficher_info()
        print("Cours enseigné: ", self.cours)
```



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
Autres modules

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

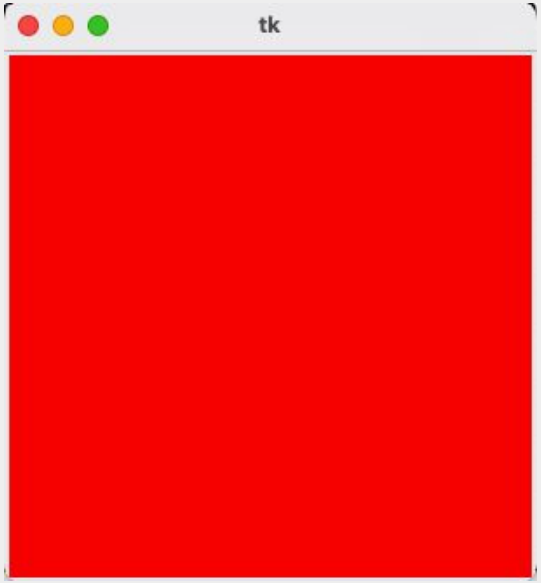
➔ Définir la couleur du background

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

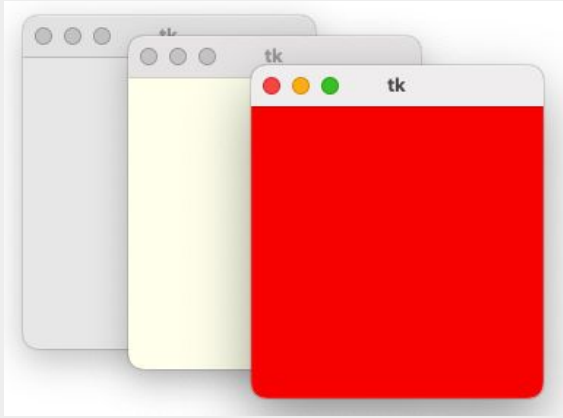


II. Fenêtres

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*.



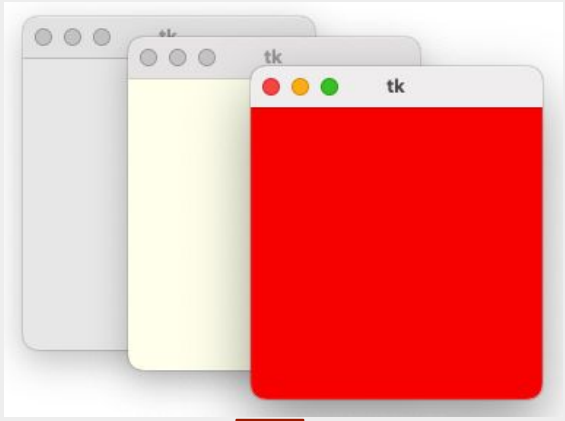

```
app = Application()
a = Toplevel(app, bg="red")
b = Toplevel(app, bg="ivory")
```

fichier GUI7.py

II. Fenêtres

1. Création de fenêtres

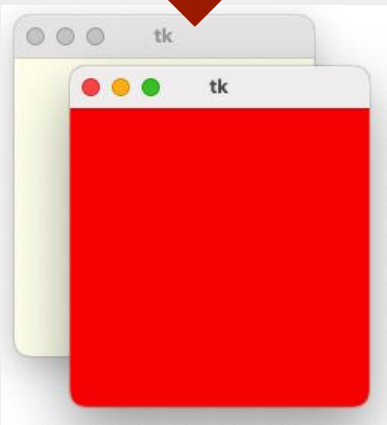
➔ Ouvrir plusieurs fenêtres



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:

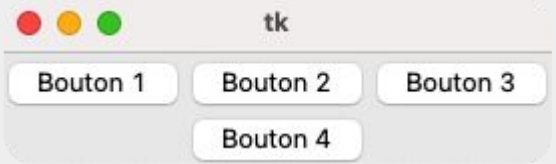
expand	**fill**	**side**
Défini sur *True* pour que le widget se développe et remplit tout espace non utilisé dans le widget parent.	Détermine si le widget remplit tout espace supplémentaire ou garde ses propres dimensions minimales: *NONE* (par défaut), *X* (remplir horizontalement), *Y* (remplir verticalement),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

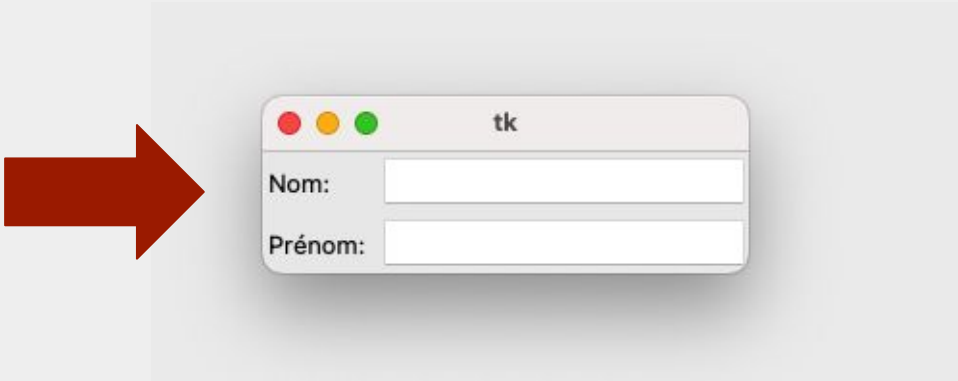
column	row	padx/pady	ipadx/ipady	rowpan/columnspan	Sticky
La colonne dans laquelle placer le widget; 0 par défaut (colonne à gauche)	La ligne dans laquelle placer le widget; par défaut la première ligne qui est encore vide.	Combien de pixels pour remplir le widget, horizontalement et verticalement, à 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

03

Les widgets et évènements

1. Quelques widgets
2. Gestion des évènements

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Frame:**

- Le widget Frame est un widget qui sert juste de conteneur, un peu comme une fenêtre Tk.
- Ce type de widget est très utile pour regrouper et organiser des interfaces contenant beaucoup de widgets.

frame.py

```
from tkinter import *

taille = 400
app = Tk()

frame = Frame(app, background="lavender")
frame.pack()

cnv = Canvas(frame, width=taille, \
    height=taille, bg="ivory")
cnv.pack(padx=20, pady=20)

btn = Button(frame, text="Coucou!")
btn.pack(pady=20)

app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Button:**

Ce widget permet de définir un bouton:

- Un bouton est construit avec le constructeur Button.
- Il faut l'inclure dans son environnement avec une méthode particulière (pack, grid ou).
- Le texte passé dans l'option text est affiché sur le bouton.
- Pour lier une action à un bouton, il faudrait lui passer une option *command*.

button.py

```
#Importer le module tkinter
from tkinter import *

#Creer l'application en tant que classe
class Application(Tk):
    def __init__(self):
        Tk.__init__(self)
    def creerBouton(self):
        texteBouton = "Quitter"
        commandeBouton = app.quit
        bouton = Button(self, text = texteBouton,\
        command = commandeBouton)
        bouton.pack(padx = 15, pady = 15)

app = Application()
app.title("Le widget Button")
app.creerBouton()
app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Button:

Pour changer la couleur du bouton:

- On utilise l'option bg.
- Sur MacOS, la couleur du bouton ne peut pas être changée.

Pour changer la couleur du texte:

- On utilise l'option fg.

```
texteBouton = "Quitter"
commandeBouton = app.quit
bouton = Button(self, text = texteBouton,\
command = commandeBouton, bg='black', fg='white')
bouton.pack(padx = 15, pady = 15)
```

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Button avec une icône:

Sur un bouton, il est usuel de placer du texte. Mais on peut placer une image aussi:

On importe d'abord une image;

Après on utilise l'option *image* au lieu de *text*.

```
logo = PhotoImage(file="red.gif")
bouton = Button(self, image=logo)
bouton.pack(padx = 15, pady = 15)
```

button-icon.py

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Canvas:**

Ce widget permet de créer une zone dans laquelle nous allons dessiner des formes diverses, ou encore insérer d'autres widgets:

- arc, bitmap, image, line, oval, polygone, rectangle, text, window,...

La syntaxe générale: *cnv = Canvas(master, options)*

bd	définir l'épaisseur de la bordure en pixels
bg	définir la couleur du background
cursor	définir le type du curseur
highlight color	la couleur du widget quand il prend le focus
width	la largeur
height	la hauteur

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Canvas:**

Quelques méthodes de la classe Canvas:

- *create_arc*(bbox, options)
- *create_image*(position, options)
- *create_line*(coords, options)
- *create_oval*(bbox, options)
- *create_text*(position, options)

canvas.py

```
from tkinter import *

app = Tk()
cnv = Canvas(app, bg="ivory", \
    height=200, width= 200)
cnv.pack()
cnv.create_oval(0, 0, 200, 200, \
    outline="red", width=10)
cnv.create_line(0, 0, 200, 200,\
    fill="black", width=10)
cnv.create_line(0, 200, 200, 0, \
    fill="black", width=10)
app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Canvas:**

Prise de focus dans un canevas:

- Si un canevas est une surface devant capturer des événements du clavier, il faut lui donner le focus pour qu'il puisse réagir aux touches de clavier.
- La méthode *focus_set()* permet à une widget de prendre le focus.

canvas-focus.py

```
from tkinter import *
from random import randrange

taille=200
app = Tk()
cnv = Canvas(app, width=taille,\
    height=taille, bg="ivory")
cnv.pack(padx=10, pady=10)
cnv.focus_set()

def dessiner(event):
    a=randrange(taille)
    b= randrange(taille)
    cnv.create_rectangle(a, b, a+20, b+2)
cnv.bind("<Key>", dessiner)

app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Label:**

- Label est un widget Tkinter standard utilisé pour afficher un texte ou une image à l'écran.
- Label ne peut afficher du texte que dans une seule police.
- Le texte affiché par ce widget peut être mis à jour à tout moment.

Syntaxe: lab = Label(master, options)

fg	définir l'épaisseur de la bordure en pixels
bg	définir la couleur du background
command	définir la fonction à appeler
font	la police et taille du texte
image	l'image à mettre sur le bouton
width	la largeur
height	la hauteur

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Label:**

Centrer un texte dans un label:

- Un label admet une option d'alignement *justify* permettant le centrage.

Image dans un label:

- Un label peut aussi porter une image au lieu de texte. Il faut remplacer l'option *text* par l'option *image*.

label.py

```
from tkinter import *

app=Tk()
mon_texte="""Je suis un texte long
qui souhaiterait être centré
sur plusieurs lignes."""
annonce=Label(app, height=5, width=50, \
    text=mon_texte, justify=CENTER, \
        bg="ivory")
annonce.pack()
app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Label:**

Utilisation d'une variable de contrôle dans un label:

- L'utilisation d'une variable de contrôle permet de mettre à jour un label suite à un évènement.
- *Exemple:* Quand l'utilisateur clique sur le bouton, un label augmente le compteur d'une unité.

string-var.py

```
from tkinter import *

app=Tk()
msg=StringVar()
entree=Entry(app, textvariable=msg)
entree.pack( padx=20, pady=10)

lbl=Label(app, textvariable=msg)
lbl.pack(padx=20, pady=10)

app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Checkbutton:**

Le widget *Checkbutton* montre du texte et une case à cocher.

Il faut créer autant de widgets *Checkbutton* que de cases à cocher.

Syntaxe: cb = CheckButton(master, options)

title	le titre du widget
active background	la couleur du background sous le curseur
active foreground	la couleur du foreground sous le curseur
bg	la couleur du background

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Checkbutton:**

```
from tkinter import *

app = Tk()
app.geometry("100x100")
var1 = IntVar()
Checkbutton(app,text="Oui",variable=var1)\
    .grid(row=0,sticky=W)
var2 = IntVar()
Checkbutton(app,text="Non",variable=var2)\
    .grid(row=1,sticky=W)

app.mainloop()
```

checkbutton.py

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Entry:**

Le widget Entry est une zone de texte qui permet à l'utilisateur de communiquer avec le programme en lui transmettant des données écrites au clavier (ou par copier-coller).

- Syntaxe: Entry(master, options)

fg	couleur du texte
font	police de texte à utiliser
command	fonction ou méthode à appeler à chaque fois que l'utilisateur modifie l'état du widget
justify	justifier le texte: LEFT, RIGHT, CENTER
show	pour masquer le message écrit par l'utilisateur (ex: saisie de mot de passe). show = « * ».
textvariable	afin de pouvoir récupérer le texte actuel, il faut définir cette option sur une instance de la classe StringVar
xscrollcommand	lier votre widget Entry à une barre de défilement

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Entry:**

```
from tkinter import *

app = Tk()
Label(app,text="Nom").grid(row=0)
Label(app,text="Prénom").grid(row=1)
e1 = Entry(app)
e2 = Entry(app)
e1.grid(row=0, column=1)
e2. grid(row=1, column=1)

app.mainloop()
```

entry1.py

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Entry:**

```
from tkinter import *

app = Tk()
entree = Entry(app)
entree.pack()
def afficher():
    print(entree.get())

bouton=Button(app, text="Afficher", \
    command=afficher)
bouton.pack(padx=50, pady=10)

app.mainloop()
```

entry2.py

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Entry:

Effacer la zone d'écriture:

- Pour effacer la zone d'édition d'une entrée, on utilise la méthode Entry.delete(first,last).
- La méthode supprime les caractères en commençant par celui de l'index *first*, jusqu'au caractère *last*. Si le deuxième argument est omis, seul le caractère à la première position est supprimé.

```
entree.delete(0, END)
```

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Listbox:

Ce widget propose une liste dont les éléments sont sélectionnables à la souris ou avec les flèches Haut et Bas du clavier.

Syntaxe: Listbox(master, options)

width	le nombre de caractères (pas de pixels)
height	le nombre d'entrées devant apparaître dans la liste
selectbackground	la couleur du fond d'une cellule sélectionnée
font	la police utilisée

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Listbox:**

listbox.py

```
from tkinter import *

app = Tk()

liste = Listbox(app)
liste.insert(1, "Bleu")
liste.insert(2, "Rouge")
liste.insert(3, "Vert")
liste.insert(4, "Jaune")
liste.insert(5, "Orange")
liste.insert(6, "Noir")

liste.pack()
app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Listbox:**

Associer une action à un widget:

listbox-action.py

```
from tkinter import *

fruits = ["Litchie", "Kiwi", "Orange", "Raisin", \
    "Banane", "Cerise"]
couleurs = ["pink", "lightgreen", "orange", \
    "purple", "yellow", "red"]
n = len(fruits)

app = Tk()

lbox = Listbox(app, width=8,height=8,\
    font="Verdana 30 bold",selectbackground="blue")
lbox.pack(padx=20,pady=20)
```

```
for item in fruits:
    lbox.insert(END, item)
    lbox.focus_set()
    pos = 1
    lbox.activate(pos)
    lbox.selection_set(pos)
for i in range(0, len(fruits), 2):
    lbox.itemconfigure(i, background="#f0f0ff")
for i in range(1, len(fruits), 2):
    lbox.itemconfigure(i, background="#fff")
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Listbox:**

Associer une action à un widget:

listbox-action.py

```
def show(event):
    index= lbox.curselection()[0]
    cnv["bg"] = couleurs[index]

lbox.bind("<<ListboxSelect>>", show)

cnv = Canvas(app, width =200, height= 200, \
    bg="ivory")
cnv.pack(padx=5,pady=5,side=RIGHT)
cnv["bg"]=couleurs[pos]

app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Radiobutton:**

Typiquement, on utilise le widget Radiobutton dans des situations de choix conduisant à une unique réponse.

Syntaxe: btn = Radiobutton(master, options)

state	DISABLED: pour griser le radiobutton et le désactiver ACTIVE: lorsque la souris est dessus NORMAL: valeur par défaut
selectcolor	la couleur du bouton radio lorsqu'il est défini (rouge par défaut)
image	Image à afficher sur le radiobutton
selectimage	afficher une image différente lorsque le bouton radio est défini
value	donnez à chaque bouton radio du groupe une valeur de type String si la variable de contrôle est StringVar, et une valeur entière si un IntVar
textvariable	définissez cette option sur cette la variable de contrôle StringVar.

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Radiobutton:**

```
from tkinter import *

def sel():
  selected = "Vous avez sélectionné : " \
    + v.get()
  label.config(text = selected)

app = Tk()
v = StringVar()
v.set("Python")
```

```
r1 = Radiobutton(app, text="Python", \
  variable=v, value="Python", command=sel)
r1.pack(anchor = W)
r2 = Radiobutton(app, text="Java", \
  variable=v, value="Java", command=sel)
r2.pack(anchor = W)
r3 = Radiobutton(app, text="PHP", \
  variable=v, value="PHP", command=sel)
r3.pack(anchor = W)

label = Label(app)
label.pack()
app.mainloop()
```

radiobutton.py

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Menubutton:**

Le widget Menubutton est la partie d'un menu déroulant qui reste à l'écran tout le temps.

Chaque MenuButton est associé à un widget Menu qui peut afficher les choix pour ce MenuButton lorsque l'utilisateur clique dessus.

direction	LEFT: pour afficher le menu à gauche du bouton RIGHT: pour afficher le menu à droite du bouton above: pour placer le menu au-dessus du bouton
bitmap	le nom de bitmap à afficher sur le menubutton
image	Image à afficher sur le menubutton
state	DISABLED: pour griser le radiobutton et le désactiver ACTIVE: lorsque la souris est dessus NORMAL: valeur par défaut
text	texte à afficher sur le menubutton.

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Menu:

Le widget Menu permet de créer toutes sortes de menus. La fonctionnalité de base permet de créer trois types de menus: pop-up, toplevel et pull-down.

activeborderwidth	la largeur d’une bordure dessinée autour d’un choix lorsqu’elle se trouve sous la souris.
postcommand	la procédure qui sera appelée chaque fois que quelqu’un affichera ce menu.

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Menu:**

menu.py

```
app = Tk()
def bonjour():
    print("Bonjour tout le monde!")
menubar = Menu(app)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Nouveau", \
    command=bonjour)
filemenu.add_command(label="Ouvrir", \
    command=bonjour)
filemenu.add_command(label="Enregistrer", \
    command=bonjour)

menubar.add_cascade(label="Fichier", \
    menu=filemenu)

app.config(menu=menubar)
app.mainloop()
```

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Spinbox:**

Ce widget dispose d'une zone de texte et deux boutons de défilement.

Ce widget permet de faire défiler des éléments dans la zone de texte en progressant dans un sens ou dans l'autre.

from_	La valeur minimale
to	La valeur maximale
values	Un tuple contenant des valeurs valides pour ce widget
textvariable	Cette option est définit sur une instance de la classe StringVar
validate	Mode de validation (NONE par défaut)
width	La largeur du widget déterminée par la taille du caractère affiché
xscrollcommand	Connecter un champ Spinbox à une barre de défilement horizontale

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Spinbox:**

```
from tkinter import *

app = Tk()

sb = Spinbox(app, from_=0, to=15)
sb.pack()

app.mainloop()
```

spinbox.py

III. Widgets et évènements

1. Quelques widgets

➔ Le widget Scrollbar:

Le widget Scrollbar fournit un contrôleur de diapositives qui est utilisé pour ajouter une barre de défilement à des widgets tels que Listbox, Text, et Canvas.

cursor	Le curseur qui apparaît lorsque la souris survole la barre de défilement.
orient	orient = HORIZONTAL pour une barre de défilement horizontale orient = VERTICAL pour une barre verticale
width	Largeur de la barre de défilement (sa dimension y si horizontale et sa dimension x si verticale)

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget Scrollbar:**

```
from tkinter import *

app = Tk()
(variable) scrollbar: Scrollbar
scrollbar = Scrollbar(app)
scrollbar.pack(side = RIGHT, fill = Y)

liste = Listbox(app,yscrollcommand=scrollba
for i in range(200):
    liste.insert(END, str(i) + " Bonjour!")

liste.pack(side = LEFT, fill = BOTH )
scrollbar.config(command = liste.yview )

app.mainloop()
```

scrollbox.py

III. Widgets et évènements

1. Quelques widgets

➔ **Le widget tkMessagebox:**

- Le widget tkMessageBox est utilisé pour afficher des boîtes de message.
- Ce module a un certain nombre de fonctions qui affichent des messages appropriés comme les fonctions: showinfo(), showwarning(), showerror(), askquestion(), askokcancel(), askyesno(), et askretryignore().

Syntaxe: tkMessageBox.FunctionName(title, message [, options])

III. Widgets et évènements

1. Quelques widgets

```
from tkinter import *
from tkinter import messagebox

app = Tk()
def msg():
    messagebox.showinfo("Info", \
        "Bonjour!")

btn = Button(app, text = "Cliquez ici!", \
    command = msg)
btn.pack()

app.mainloop()
```

tkmessagebox.py

III. Widgets et évènements

1. Quelques widgets

➔ **Modifier les options d'un widget:**

Il existe deux syntaxes pour changer une option:

- *widget_name["option"] = new_value*
- *widget_name.configure(option = new_value)*

Il est aussi possible de modifier l'état de manière dynamique en utilisant les variables de contrôle comme StringVar et IntVar.

III. Widgets et évènements

2. Gestion des évènements

➔ Capturer des évènements:

Une interface Tkinter est à l'écoute de certains événements liés à la souris ou au clavier.

Exemples: appuyer ou relâcher une touche du clavier, déplacer le curseur de la souris, cliquer ou relâcher le bouton de la souris.

Il est possible de lier (bind en anglais) un événement à un widget ou fenêtre et aussi définir la fonction à exécuter en utilisant la méthode *bind()*.

III. Widgets et évènements

2. Gestion des évènements

➔ Capturer des évènements:

Exemple de l'utilisation de la méthode bind():

```
from tkinter import *

def f(event):
    t=event.keysym
    print("Touche pressée :", t)
def g(event):
    x=event.x
    y=event.y
    print("Position :", x, y)

app = Tk()
app.bind("<Key>", f)
app.bind("<Motion>",g)
app.mainloop()
```

bind.py

III. Widgets et évènements

2. Gestion des évènements

➔ Évènements du clavier:

Un code d'événement est une chaîne de la forme "<*event*>" où *event* est une chaîne de caractères.

Quelques évènements du clavier:

- Flèches : Left, Right, Up, Down
- ESPACE : space
- Touche ENTRÉE : Return ou, sur le pavé numérique, KP_Enter
- Touche ECHAP : Escape

L'appui sur une touche de caractère, par exemple **a/A**, se nomme **KeyPress-a/KeyPress-A**. On peut abréger le code à seulement **a/A**.

Pour le relâchement d'une touche de caractère: **KeyRelease-a.**

Pour combiner plusieurs touches (ex: ALT+CTRL-a), l'évènement sera nommé: <Control-Shift-KeyPress-a>

III. Widgets et évènements

2. Gestion des évènements

➔ Événements de la souris:

Un widget peut capturer des actions de la souris. Voici un résumé des principaux actions liés à la souris :

Button-1	Clic gauche
Button-3	Clic bouton droit
Button-2	Clic bouton central
Button-4	Molette vers le haut
Button-5	Molette vers le bas
ButtonRelease	Relâchement d’un bouton
Motion	Déplacement du curseur de la souris

III. Widgets et évènements

2. Gestion des évènements

➔ Exemple:

mouse-event.py

```
from tkinter import *

LARGEUR = 480
HAUTEUR = 320

def clic(event):
    X = event.x
    Y = event.y
    r = 20
    cnv.create_rectangle(X-r, Y-r, X+r, Y+r,\
        outline = 'black',fill = 'green')

def effacer():
    cnv.delete(ALL)

app= Tk()
```

III. Widgets et évènements

2. Gestion des évènements

➔ Exemple:

mouse-event.py

```
cnv = Canvas(app, width = LARGEUR, \
    height = HAUTEUR, bg = 'white')
cnv.pack(padx = 5, pady = 5)

cnv.bind('<Button-1>', clic)
cnv.pack(padx =5, pady =5)

Button(app, text = 'Effacer', \
    command = effacer).\
        pack(side = LEFT,padx = 5,pady = 5)
Button(app, text = 'Quitter', \
    command = app.destroy).pack()

app.mainloop()
```

III. Widgets et évènements

2. Gestion des évènements

➔ Fonction lambda:

La fonction lambda est une petite fonction contenant qu'une seule expression.

Elle peut agir sous anonymat parce qu'elle ne nécessite aucun nom.

Elles peuvent comporter n'importe quel nombre d'arguments mais une seule expression.

Voici la syntaxe de la fonction lambda en Python :

lambda x : y (**x** les paramètres / **y** le corps de la fonction)

Différence entre la fonction lambda et la fonction régulière:

lambda x: x+3

def add(x):

return x+3

III. Widgets et évènements

2. Gestion des évènements

➔ Fonctions lambda (exemple 1):

- Utiliser une fonction lambda avec un bouton:

Button(master, text = “Bonjour”, command = **lambda : fct_name(arg1, arg2, ….)**)

(voir fichier lambda_fct.py)

III. Widgets et évènements

2. Gestion des évènements

➔ Fonctions lambda (exemple 2):

- Utiliser une fonction lambda avec la méthode *bind()*

```
fen = Tk()
fen.bind("<Key>", lambda event, args: fct_name(args))
fen.bind("<Key>", lambda event, args: fct_name(event, args))
```

(voir fichier lambda_fct1.py)

III. Widgets et évènements

2. Gestion des évènements

➔ Changer l'icône d'une fenêtre:

Il existe différentes méthodes pour changer l'icône d'une fenêtre Tkinter:

- ➔ fenetre.*iconbitmap*("photo path") – le bitmap doit être de type .ico
- ➔ fenetre.*iconphoto*(default = False, photo)
- ➔ fenetre.*tk.call*('wm', 'iconphoto', fenetre._w, photo)

Exemples: iconphoto.py et iconcall.py

III. Widgets et évènements

2. Gestion des évènements

➔ Le module ttk:

ttk est une extension de Tk qui fournit un accès au jeu de style pour les widgets Tk.

Ce module contient des widgets identiques que Tk : Button, Checkbutton, Label, Entry, etc… mais contient 6 autres widgets qui ne sont pas disponibles sur Tk:

- Combobox, Notebook, Progressbar, Separator, Sizegrip et Treeview.

Exemples: combobox.py - progressbar.py - sizegrip.py - treeview.py

III. Widgets et évènements

2. Gestion des évènements

➔ Le module ttk:

Modifier le thème de la fenêtre:

- Instancier la classe *Style*:

  style = ttk.*Style*(master)

- Définir le thème à utiliser avec *theme_use()*:

  style.*theme_use*(‘nom du thème’)

form.py

III. Widgets et évènements

2. Gestion des évènements

➔ Le module ttk:

Modifier le thème de la fenêtre:

- La méthode *theme_names*()

renvoie la liste des thèmes disponibles.

theme.py

III. Widgets et évènements

2. Gestion des évènements

➔ Basculer entre les frames:

La méthode *tkraise*() de la classe Frame permet de placer une frame sur d'autres.

Il est aussi possible d'utiliser pack_forget() , grid_forget() ou place_forget().

frame_switch.py

04

Pour aller plus loin

1. Animation
2. Audios sous Tkinter
3. Autres modules

IV. Pour aller plus loin

1. Animation:

➔ La méthode After():

- La méthode *after()* est une méthode pouvant être utilisée avec n'importe quel widget afin d'appeler avec un certain délai (ou même périodiquement) l'exécution d'une fonction qui se charge d'animer.

  cnv.*after*(ms, action, args)

- Cette méthode renvoie un identifiant de la tâche qui va être relancée (ex: *id_anim*).
- Pour annuler la tâche on utilise la méthode *after_cancel():*

  cnv.*after_cancel*(id_anim)

Exemples: after1.py - after2.py - after3.py - cancelafter.py

IV. Pour aller plus loin

2. Audio:

- Tkinter ne prend pas en charge la diffusion du flux audio.
- Il est possible d'utiliser d'autres bibliothèques pour faire émettre du son.
- Exemples de bibliothèques: Pygame, Pyglet, Winsound, Playsound.
  - ➔ *Pygame*: compatible avec le format .wav
  - ➔ *Pyglet*: compatible avec le format .wav
  - ➔ *Winsound:* marche sur windows seulement
  - ➔ *Playsound:* compatible avec windows, mac OS et Linux (mp3, wav, …)

Exemples: pyglet_sound.py / play_sound.py / pygame_sound.py

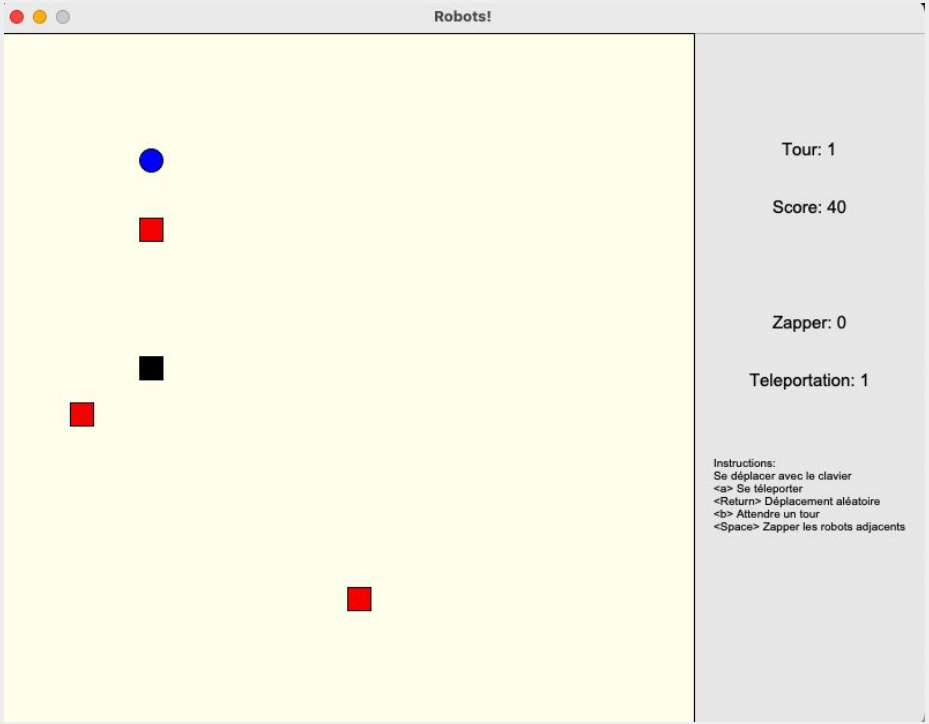
IV. Pour aller plus loin

3. Autres modules:

- Il existe d'autres bibliothèques qui permettent de créer des interfaces graphique avec Python:
  - ➔ QtPy
  - ➔ GTK
  - ➔ Pygame
  - ➔ Pyglet
  - ➔ Arcade

IV. Pour aller plus loin

ttt.py



escape.py

Programmation graphique avec Qt

1. Introduction
2. Fenêtre et widgets
3. Mise en forme
4. Evénements

I. Introduction

1. Le module Qt:

- **Qt:**

  codé en C++, contient plusieurs widgets permettant la création des GUIs.

- **PySide** et **PyQt**:

  deux frameworks permettant de faire le binding entre Qt et Python

- **QtPy**:

  une couche d'abstraction permettant de créer des applications en utilisant un appel unique à Pyside ou PyQt.

I. Introduction

2. Binding de Qt:

Le binding de Qt est possible pour les langages suivants: Python, Ring, Go, Rust, PHP, and Java™.

Pour utiliser le module PyQt, il faut l'installer sur votre machine. Il est compatible avec:

- Mac Os
- Linux
- Windows
- Android
- iOS

II. Fenêtre et widgets

1. Création d'une fenêtre:

- ➔ Pour créer une fenêtre, on utilise la classe *QWidget*.
- ➔ Un widget qui n'est pas intégré dans un widget parent est appelé une fenêtre.
- ➔ La classe *QApplication* gère les ressources de l'application et est requise pour n'importe quelle application graphique.
- ➔ La méthode *show()* de la classe *QWidget* rend un widget visible.
- ➔ La méthode *exec_()* de la classe *QApplication* permet d'exécuter l'application.

fenetre.py

```
import sys
from PyQt5.QtWidgets import *

app = QApplication.instance()
app = QApplication(sys.argv)

fen = QWidget()
fen.show()
app.exec_()
```

II. Fenêtre et widgets

1. Création d'une fenêtre:

setWindowTitle(text)	définir le titre de la fenêtre
move(x,y)	fixer la position de la fenêtre
resize(width,height)	fixer la taille de la fenêtre
setStyleSheet("background-color: color;")	changer la couleur de la fenêtre
setMinimumWidth()	fixer la largeur minimale
setMinimumHeight()	fixer la hauteur minimale

fen_personalisee.py

```
import sys
from PyQt5.QtWidgets import *

app = QApplication.instance()
app = QApplication(sys.argv)

fen = QWidget()
fen.setWindowTitle("Ma fenêtre")
fen.resize(400,400)
fen.move(300,150)

fen.show()
app.exec_()
```

II. Fenêtre et widgets

2. Création des widgets:

➔ Le widget QLabel:

Ce widget permet d'afficher un texte non éditable par l'utilisateur, mais que le programme peut faire évoluer.

- Instanciation de la classe:
  mon_label = *QLabel*("texte initiale")
- Modification du texte:
  mon_label.*setText*("Mon texte")
- Changer la police du texte:
  mon_label.*setStyleSheet*("border: 1px solid black;")
- Ajuster automatiquement la taille du label:
  mon_label.*adjustSize*()

qlabel.py

```
import sys
from PyQt5.QtWidgets import *

app = QApplication.instance()
app = QApplication(sys.argv)

mon_label = QLabel("Mon widget QLabel")

mon_label.show()
app.exec_()
```

II. Fenêtre et widgets

2. Création des widgets:

➔ Le widget QPushButton:

Le constructeur de la classe *QPushButton* permet de créer un bouton crée un bouton.
On lui communique le texte qu'on souhaite voir figurer à l'intérieur du bouton:

- Instanciation de la classe:

  mon_bouton = *Button*("mon texte")
- Connecter le signal "clicked" avec l'appel d'une fonction:

  mon_bouton.*clicked.connect*(ma_fonction)
- Changer le texte du bouton: *setText*()

qbutton.py

```
import sys
from PyQt5.QtWidgets import *

def appui_bouton():
    print("Appui sur le bouton")

app = QApplication.instance()
app = QApplication(sys.argv)

mon_bouton = QPushButton("Mon bouton Qt")
mon_bouton.clicked.connect(appui_bouton)

mon_bouton.show()
app.exec_()
```

II. Fenêtre et widgets

2. Création des widgets:

➔ Le widget QLineEdit:

En instanciant la classe *QLineEdit* on obtient une zone de texte éditable dans laquelle l'utilisateur peut entrer ou modifier un texte.

On peut connaître à tout moment le texte contenu dans un champ de texte à l'aide de la méthode *text()*.

qline.py

```
import sys
from PyQt5.QtWidgets import *

app = QApplication.instance()
app = QApplication(sys.argv)

champ = QLineEdit("Tapez votre message")

champ.show()
app.exec_()
```

II. Fenêtre et widgets

2. Création des widgets:

➔ Le widget QCheckBox:

La classe *QCheckBox* permet de créer des cases à cocher.

- La méthode *checkState():* permet de connaître l'état d'une case à cocher.
- La méthode *setCheckState():* permet de changer l'état d'une case à cocher.

Pour déclencher un événement à chaque changement d'état, on connecte le signal stateChanged avec l'appel d'une fonction.

qcheckbox.py

```
import sys
from PyQt5.QtWidgets import *

def etat_change():
    print("action sur la case")

app = QApplication.instance()
app = QApplication(sys.argv)

case = QCheckBox("Ma case à cocher")
case.stateChanged.connect(etat_change)

case.show()
app.exec_()
```

III. Mise en forme

1. Le gestionnaire de mise en forme

➔ Il existe différents gestionnaires de mise en forme:

- *QVBoxLayout :* positionne les composants les uns sous les autres verticalement.
- *QHBoxLayout :* positionne les composants les uns sous les autres horizontalement.
- *QGridLayout :* positionne les composants dans une grille.
- *QStackedLayout :* place les widget un sur l'autre et donc un seul widget est visible à un moment donné.

III. Mise en forme

1. Le gestionnaire de mise en forme

- Il existe différents gestionnaires de mise en forme:
  - ***QVBoxLayout :*** positionne les composants les uns sous les autres verticalement.
  - *QHBoxLayout :* positionne les composants les uns sous les autres horizontalement.
  - *QGridLayout :* positionne les composants dans une grille.
  - *QStackedLayout :* place les widget un sur l'autre et donc un seul widget est visible à un moment donné.

- La classe QVBoxLayout possède une méthode *addWidget()* pour ajouter les composants à gérer.
- Il faut indiquer à la fenêtre qu'elle doit utiliser ce gestionnaire de mise en forme grâce à la méthode *setLayout()* de la classe QWidget.

(exemple: qvlayout.py)

III. Mise en forme

1. Le gestionnaire de mise en forme

- Il existe différents gestionnaires de mise en forme:
  - *QVBoxLayout :* positionne les composants les uns sous les autres verticalement.
  - ***QHBoxLayout :*** positionne les composants les uns sous les autres horizontalement.
  - *QGridLayout :* positionne les composants dans une grille.
  - *QStackedLayout :* place les widget un sur l'autre et donc un seul widget est visible à un moment donné.

- La classe QHBoxLayout possède une méthode *addWidget()* pour ajouter les composants à gérer.
- Il faut indiquer à la fenêtre qu'elle doit utiliser ce gestionnaire de mise en forme grâce à la méthode *setLayout()* de la classe QWidget.

(exemple: qhlayout.py)

III. Mise en forme

1. Le gestionnaire de mise en forme

- Il existe différents gestionnaires de mise en forme:
  - *QVBoxLayout :* positionne les composants les uns sous les autres verticalement.
  - *QHBoxLayout :* positionne les composants les uns sous les autres horizontalement.
  - ***QGridLayout :*** positionne les composants dans une grille.
  - *QStackedLayout :* place les widget un sur l'autre et donc un seul widget est visible à un moment donné.
- Il possède une méthode *addWidget()* pour ajouter les composants à gérer.
  addWidget(color, row, column)
- Il faut indiquer à la fenêtre qu'elle doit utiliser ce gestionnaire de mise en forme grâce à la méthode *setLayout()* de la classe QWidget.

(exemple: qgrid.py)

IV. Evénements

1. Événements de la souris

- ➔ La méthode *mousePressEvent* est un gestionnaire d'événement qui est automatiquement appelé lors de l'appui sur un bouton de la souris.
- ➔ Les arguments transmis aux méthodes qui gèrent les événements de la souris sont des objets de type *QMouseEvent*.
- ➔ Cet objet contient une information concernant le bouton qui a été appuyé.

mousePressEvent	appui sur un bouton de la souris
mouseReleaseEvent	relâchement d'un bouton de la souris
mouseDoubleClickEvent	double clic sur un bouton de la souris
mouseMoveEvent	mouvement de la souris (par défaut, quand un bouton est appuyé)

IV. Evénements

1. Événements de la souris

- ➔ Pour récupérer le bouton appuyé, on utilise la méthode *button().*
- ➔ Pour récupérer les coordonnées du curseur au moment de l'appui, on utilise les deux méthodes *x()* et *y()*.
- ➔ Dans le cas du gestionnaire *mouseMoveEvent*, le suivi des mouvements de la souris doit être activé, on utilise la méthode *setMouseTracking*(True)

2. Évènements du clavier

- ➔ *keyPressEvent* et *keyReleaseEvent* sont deux méthodes appelées à chaque appui et relâchement d'une touche du clavier.

events.py

```
import sys
from PyQt5.QtWidgets import *

class Fenetre(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.setWindowTitle("Ma fenetre")

    def mousePressEvent(self, event):
        print("appui souris")

app = QApplication.instance()
app = QApplication(sys.argv)
fen = Fenetre()
fen.show()
app.exec_()
```