Constructions d'Interfaces Graphiques

TkInter

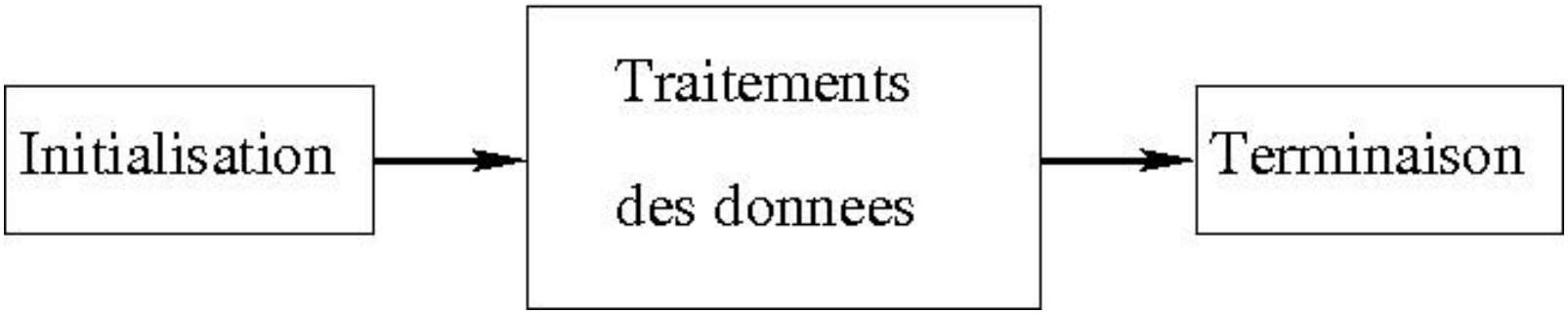
Alexis NEDELEC

LISYC EA 3883 UBO-ENIB-ENSIETA
Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Programmation classique

3 phases séquentielles

1. initialisation
   - importer les modules externes
   - ouverture de fichiers
   - connexions serveurs SGBD, Internet ..
2. traitements
   - affichages, calculs, modifications des données
3. terminaison
   - sortir “proprement” de l’application



Programmation événementielle

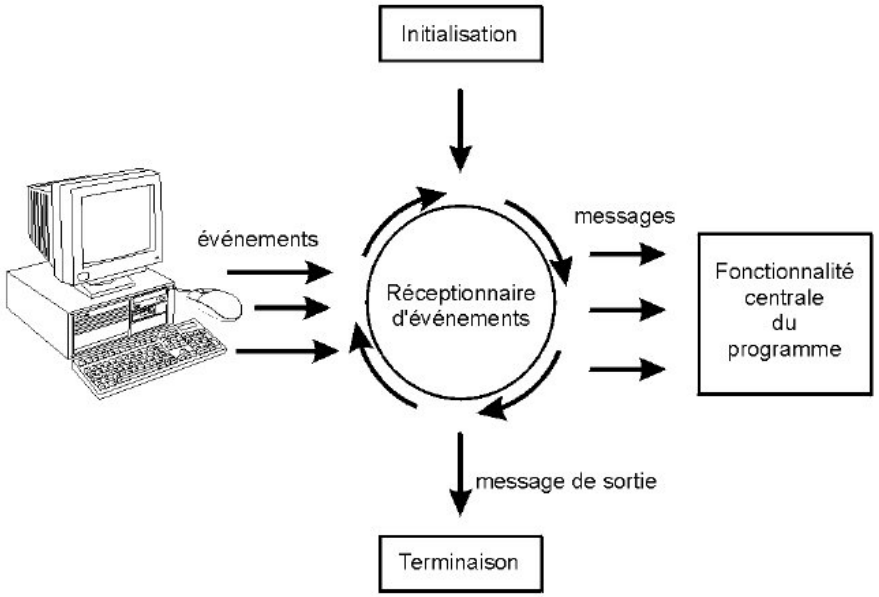
3 phases non-séquentielles

1. initialisation
   - création de composants graphiques
   - liaisons composant-événement-action
2. traitements
   - création des fonctions correspondant aux actions
   - attente d'événement lié à l'interaction utilisateur-composant
   - éxécution de l'action lié à l'apparition de l'événement
3. terminaison
   - sortir "proprement" de l'application

Programmation événementielle

Gestionnaire d'événements : event-driven programming

- à l'écoute des périphériques (clavier, souris ...)
- réaction suivant l'arrivée d'un événement
- événement détecté suivant l'action d'un utilisateur
- envoi d'un message au programme
- éxécution d'un bloc de code (fonction) spécifique



Python et TkInter

Programme classique

faible interaction (textuelle) séquentielle avec l'utilisateur

```
{logname@hostname} python
...
>>> print "hello world"
hello world
>>> exit(0)
{logname@hostname}
```

Programmation événementielle

interaction forte et non-séquentielle avec l'utilisateur

```
{logname@hostname} python hello.py
```

Hello World

Mon premier programme : hello.py

```
from Tkinter import Tk,Label,Button
mw=Tk()
labelHello=Label(mw, text="Hello World !",fg="blue")
labelHello.pack()
buttonQuit=Button(mw, text="Goodbye World", fg="red",\
                  command=mw.destroy)
buttonQuit.pack()
mw.mainloop()
exit(0)
```

Hello World

Création de la fenêtre principale et de composants

- `mw=Tk()`
- `labelHello=Label(mw, ...)`
- `buttonQuit=Button(mw, ...)`

Interaction sur le composant

- `buttonQuit=Button( ..., command=mw.destroy)`

Affichage: positionnement des composants

- `labelHello.pack()`, `buttonQuit.pack()`

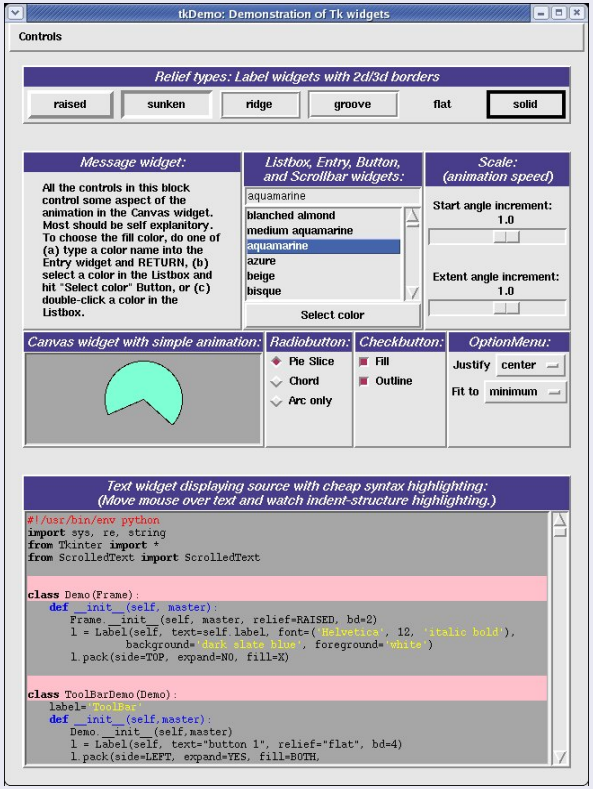
Boucle d'événements : en fin de programme

- `mw.mainloop()`

Module Tkinter

Composants graphiques de base

Tk Interface : adaptation de la bibliothèque Tcl/Tk



Composants graphiques de base

Widgets : **W**indow ga**dgets**

Fonctionnalités des widgets, composants d'IHM

- affichage d'informations (label, message...)
- composants d'interaction (button, scale ...)
- zone d'affichage, saisie de dessin, texte (canvas, entry ...)
- conteneur de composants (frame)
- fenêtres secondaires de l'application (toplevel)

Composants graphiques de base

TkInter : fenêtres, conteneurs

- `Toplevel` : fenêtre secondaire de l'application
- `Canvas` : afficher, placer des "éléments" graphiques
- `Frame` : surface rectangulaire pour contenir des widgets
- `Scrollbar`: barre de défilement à associer à un widget

TkInter : gestion de textes

- `Label`: afficher un texte, une image
- `Message`: variante de label pour des textes plus importants
- `Text`: afficher du texte, des images
- `Entry`: champ de saisie de texte

Composants graphiques de base

Tkinter: gestion de listes

- `Listbox`: liste d'items sélectionnables
- `Menu`: barres de menus, menus déroulants, surgissants

Tkinter: composants d'interactions

- `Menubutton`: item de sélection d'action dans un menu
- `Button`: associer une interaction utilisateur
- `Checkbutton`: visualiser l'état de sélection
- `Radiobutton`: visualiser une sélection exclusive
- `Scale`: visualiser les valeurs de variables

Etapes de programmation

TkInter : Structuration d'un programme

```
# ------- Initialisation ------------------------
from Tkinter import Tk,Label,Button
# ------- Composants graphiques -------------------
mw=Tk()
labelHello=Label(mw, text="Hello World !",fg="blue")
buttonQuit=Button(mw, text="Goodbye World", fg="red",\
                  command=mw.destroy)
# ------- Positionnement des composants -----------
labelHello.pack()
buttonQuit.pack()
# ------- Definition des interactions -------------
# ------- Gestion des événements  -----------------
mw.mainloop()
exit(0)
```

Gestion d'événements

Interaction par défaut : option `command`

- en cas de "click gauche" exécuter la fonction associée

Paramétrer l'interaction utilisateur : méthode `bind()`

- lier (bind) l'événement au comportement d'un composant

gestion des interactions

```
# ------- Definition des interactions --------------
    def sortie(event) :
            mw.destroy()
# ------- Gestion des evenements  ------------------
buttonQuit.bind("<Button-1>", sortie)
mw.mainloop()
exit(0)
```

Gestion d'événements

Types d'événements

représentation générale d'un événement :

- `<Modifier-EventType-ButtonNumberOrKeyName>`

Exemples

- `<Control-KeyPress-A>` (`<Control-Shift-KeyPress-a>`)
- `<KeyPress>`, `<KeyRelease>`
- `<Button-1>`, `<Motion>`, `<ButtonRelease>`

Principaux types

- `Expose` : exposition de fenêtre, composants
- `Enter`, `Leave` : pointeur de souris entre, sort du composant
- `Configure` : l'utilisateur modifie la fenêtre
- ...

Interaction Utilisateur

Informations utiles à l'interaction

- donnnées liées aux périphériques de l'utilisateur
  - argument `event`
- données liés au composant graphique d'interaction
  - `configure()` : fixer des valeurs aux options de widget
  - `cget()` : récupérer une valeur d'option

Affichage des coordonnées du pointeur de souris

```
def mouseLocation(event):
    labelPosition.configure(text = "Position X =" \
                                    + str(event.x) \
                                    + ", Y =" \
                                    + str(event.y))
#    print event.widget.cget("width")
```

Interaction utilisateur

Affichage des coordonnées du pointeur de souris

```
canvas = Canvas(mw, \
                width =200, height =150, \
                bg="light yellow")
labelPosition = Label(mw,text="Mouse Location")
canvas.bind("<Motion>", mouseLocation)
```

Interaction utilisateur

Traitement des données Utilisateur

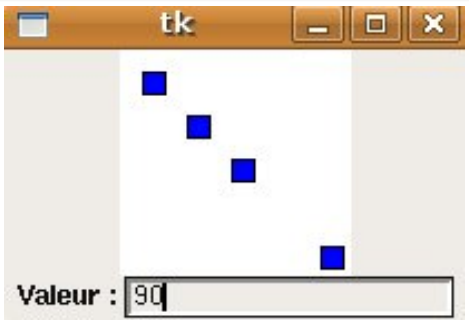
```
from Tkinter import Tk,Entry,Label
mw = Tk()
entry = Entry(mw)
label = Label(mw)
entry.pack()
label.pack()

def evaluer(event):
    label.configure(text =
                    "Resultat = "
                    + str(eval(entry.get()))
                    )
entry.bind("<Return>", evaluer)
mw.mainloop()
```

Interaction entre composants

event_generate() : émission d'événements

```
from Tkinter import *
root = Tk()
canvas=Canvas(root,width=100,height=200,bg='white',bd=1)
label= Label(root, text = 'Valeur :')
entry = Entry(root)
canvas.pack()
label.pack(side=LEFT)
entry.pack(side=LEFT)
```



Interaction entre composants

event_generate() : émission d'événements

```
def affiche(event):
    print "draw"
    x=int(entry.get())
    canvas.create_rectangle(x,x,x+10,x+10,fill="blue")
def setValue(event):
    print "setValue"
    canvas.event_generate('<Control-Z>')

root.bind('<Control-Z>', affiche)
entry.bind('<Return>', setValue)
root.mainloop()
```

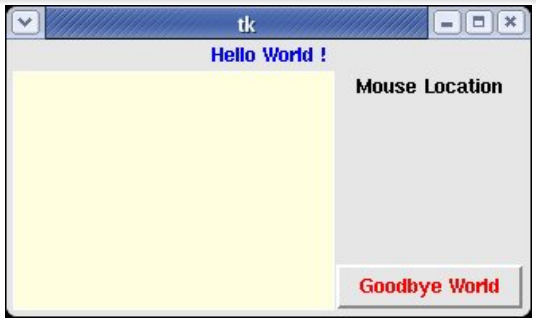
Positionnement de composants

TkInter : Layout manager

- `pack()` : "coller" les widgets par leur côté
- `grid()` : agencer en ligne/colonne
- `place()`: positionner géométriquement

`pack()` : exemple de positionnement

```
labelHello.pack()
canvas.pack(side=LEFT)
labelPosition.pack(side=TOP)
buttonQuit.pack(side=BOTTOM)
```



Positionnement de composants

Frame : agencement de composants graphiques

```
frameCanvas = Frame(mw,bg="yellow")
canvas = Canvas(frameCanvas, width=200, height=150,\
                bg="light yellow")
labelPosition = Label(frameCanvas,text="Mouse Location")
labelHello.pack()
frameCanvas.pack(fill="both",expand=1)
buttonQuit.pack()
canvas.pack(fill="both",expand=1)
labelPosition.pack()
```

Positionnement de composants

grid() : exemple de positionnement

```
labelNom = Label(mw, text = 'Nom :')
labelPrenom = Label(mw, text = 'Prenom :')
entryNom = Entry(mw)
entryPrenom = Entry(mw)
labelNom.grid(row=0)
labelPrenom.grid(row=1)
entryNom.grid(row=0,column=1)
entryPrenom.grid(row=1,column=1)
```

Positionnement de composants

place() : exemple de positionnement

```
mw.title("Layout Manager : Place")
msg = Message(mw, text="Place : \n
                    options de positionnement de widgets",
                    justify="center",
                    bg="yellow", relief="ridge")
okButton=Button(mw,text="OK")

msg.place(relx=0.5,rely=0.5,
          relwidth=0.75,relheight=0.50,
          anchor="center")
okButton.place(relx=0.5,rely=1.05,
               in_=msg,
               anchor="n")
```

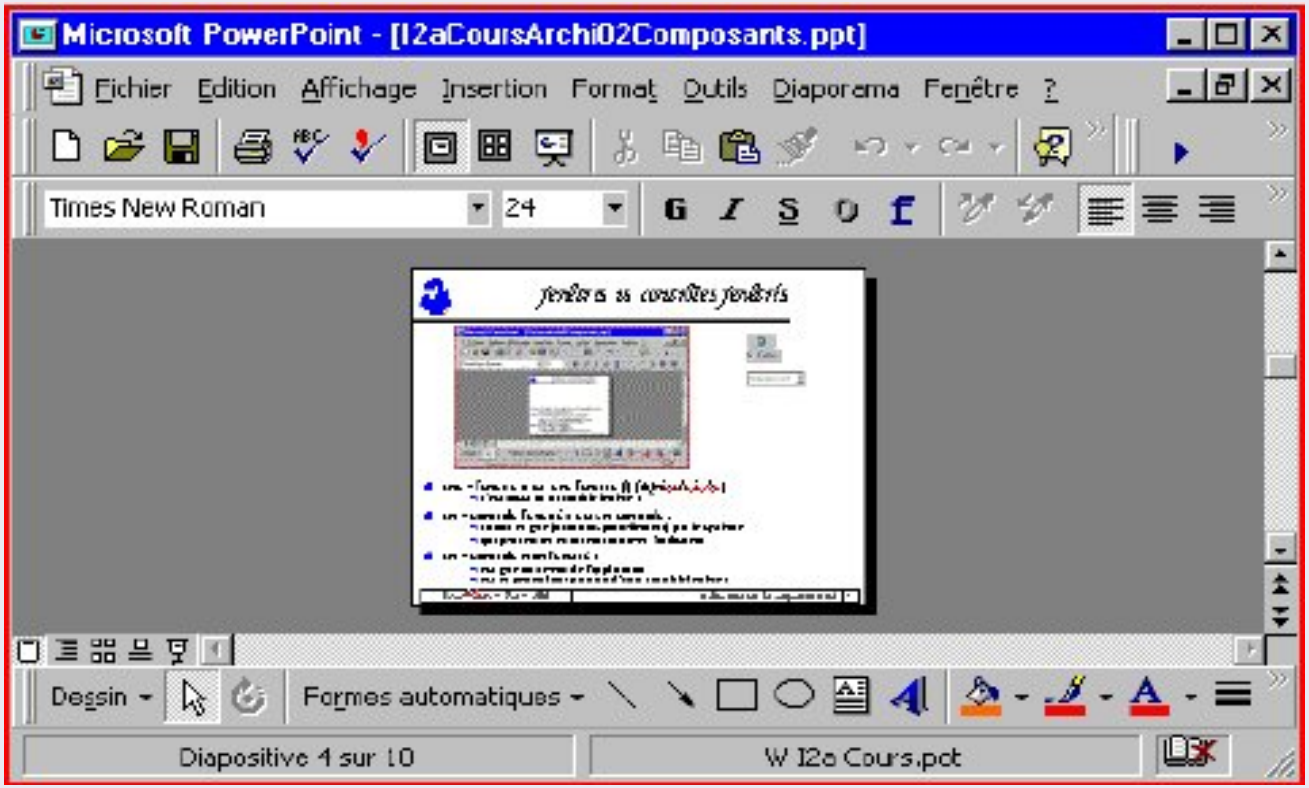
Positionnement de composants

place() : exemple de positionnement

```
def move(event):
    print "Deplacement sur l'ecran X =" \
          + str(event.x_root) \
          + ", Y =" + str(event.y_root)
def position(event):
    print "Position sur le composant X =" \
          + str(event.x) + ", Y =" + str(event.y)
msg.bind("<Motion>", move)
msg.bind("<ButtonPress>", position)
```

Création d'IHM

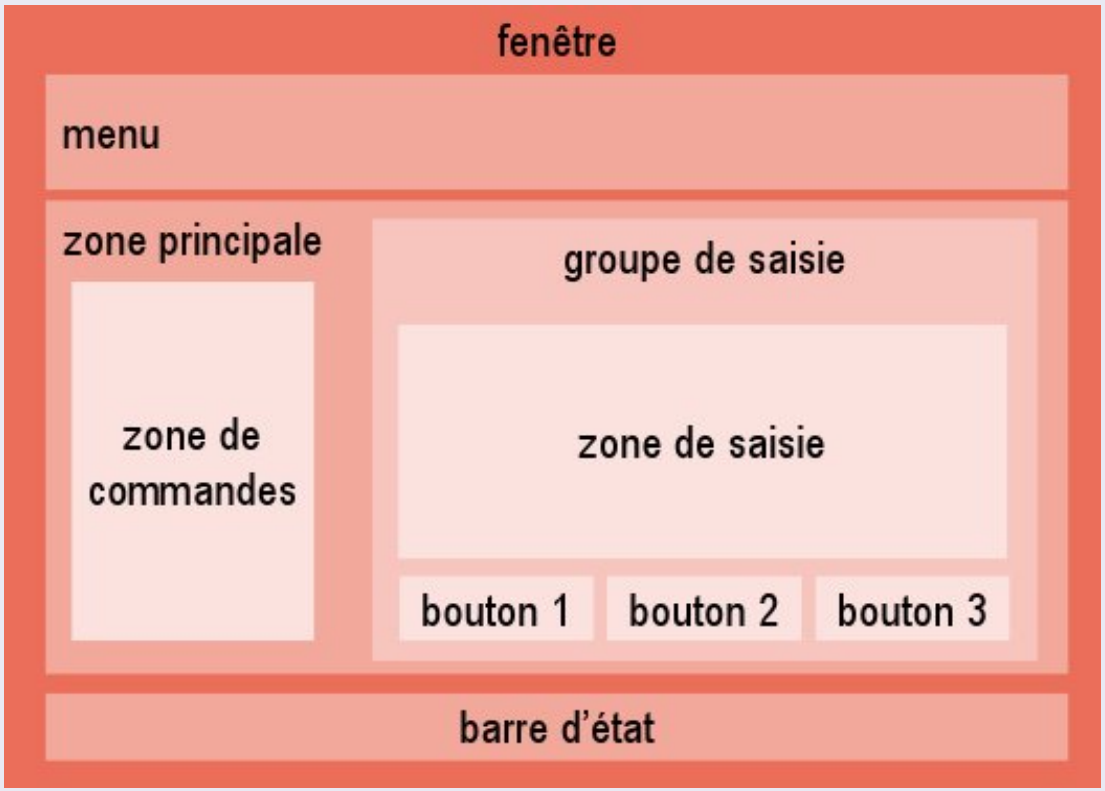
Exemple "classique"



(cf Didier Vaudène : "un abc des IHM")

Création d'IHM

Organisation hiérarchique "classique"



(cf Stéphanie Jean-Daubias : "Programmation événementielle")

Création d'IHM

Types de fenêtres

- définition des fenêtres de l'application
  - primaire, secondaires
  - boîte de dialogues, de messages
- organisation de leur contenu
- logique d'enchaînement des fenêtres

Composants de fenêtre

- barre d'actions (menu)
- région client, menus surgissants
- barre d'outils
- barre d'états

Fenêtre principale

MainWindow: Fenêtre de base

```
class MainWindow(Tk):
    def __init__(self, width=100,height=100,bg="white"):
        Tk.__init__(self)
        self.title('Editeur Graphique')
        self.canvas =Canvas(self,width=width-20,
                            height=height-20, bg=bg)
        self.libelle =Label(text ="Serious Game",
                            font="Helvetica 14 bold")
        self.canvas.pack()
        self.libelle.pack()
```

Barre de menu

MainWindow : Barre de menu

```
class MainWindow(Frame):
    def __init__(self, width=200,height=200,bg="white"):
        Frame.__init__(self)
        self.master.title('Editeur Graphique')
        ...
        self.pack()
        self.libelle.pack()
        self.canvas.pack()
```

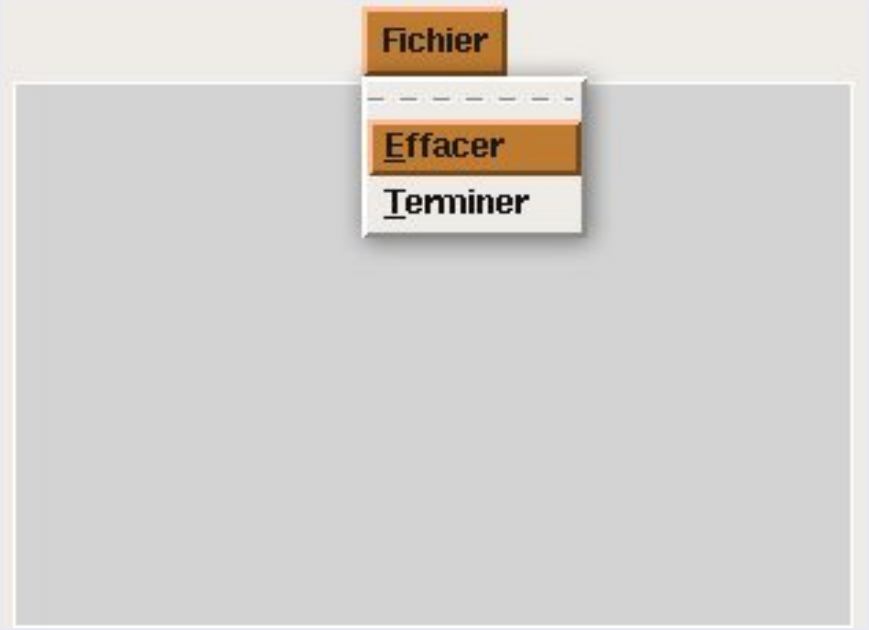
Barre de menu

MenuBar : Menu déroulant

```
class MenuBar(Frame):
    def __init__(self,boss=None):
        Frame.__init__(self,borderwidth=2)
        mbuttonFile = Menubutton(self,text='Fichier')
        mbuttonFile.pack(side=LEFT)
        menuFile=Menu(mbuttonFile)
        menuFile.add_command(label='Effacer',
                             command=boss.effacer)
        menuFile.add_command(label='Terminer',
                             command=boss.quit)
        mbuttonFile.configure(menu=menuFile)
```

Barre de menu

MenuBar : Menu déroulant



Zone Client

ScrolledCanvas : zone défilante

```
class ScrolledCanvas(Frame):
    """Zone Client"""
    def __init__(self, boss,
                 width=100,height=100,bg="white",
                 scrollregion =(0,0,300,300)):
        Frame.__init__(self, boss)
        self.canvas=Canvas(self,
                           width=width-20,
                           height=height-20,bg=bg,
                           scrollregion=scrollregion)
        self.canvas.grid(row=0,column=0)
```

Zone Client

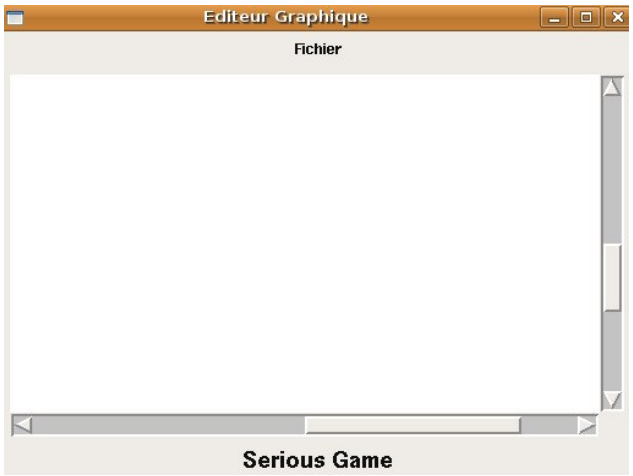
ScrolledCanvas : scrollbars

```
scv=Scrollbar(self,orient=VERTICAL,
              command =self.canvas.yview)
sch=Scrollbar(self,orient=HORIZONTAL,
              command=self.canvas.xview)
self.canvas.configure(xscrollcommand=sch.set,
                      yscrollcommand=scv.set)
scv.grid(row=0,column=1,sticky=NS)
sch.grid(row=1,colum=0,sticky=EW)
self.bind("<Configure>", self.retailler)
self.started =False
```

Zone Client

ScrolledCanvas : redimensionnement

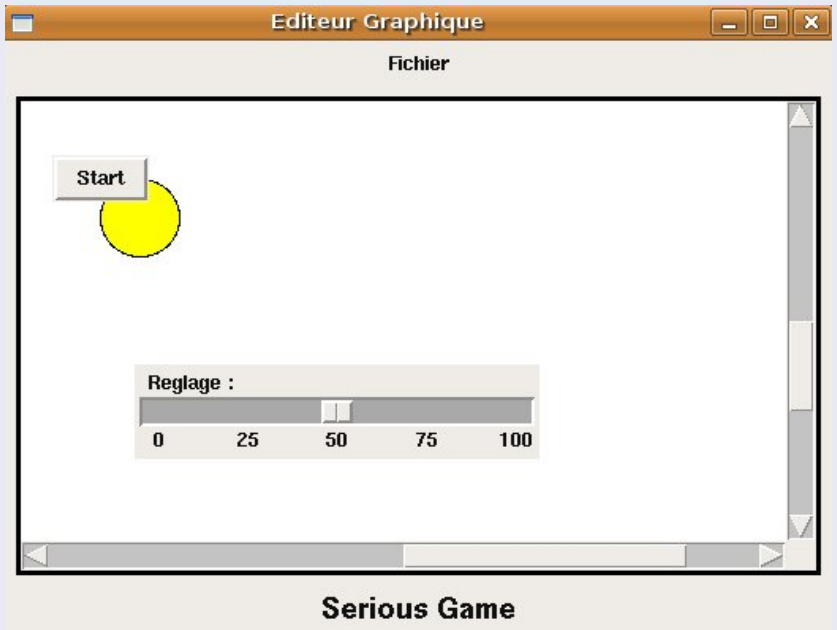
```
def retailler(self,event):
    if not self.started:
        self.started=True
        return
    larg=self.winfo_width()-20,
    haut=self.winfo_height()-20
    self.canvas.config(width=larg,height=haut)
```



Application : lancement

test de l'application

```
if __name__ =="__main__":
    MainWindow().mainloop()
```



Application : initialisation

MainWindow : __init__(self)

```
...
self.x,self.y=50,50
self.buttonStart=Button(self.canvas,
                        text="Start",
                        command=self.start)
self.scaleCercle=Scale(self.canvas,
                       length=250,orient=HORIZONTAL,
                       label='Echelle :',
                       troughcolor ='dark grey',
                       sliderlength =20,
                       showvalue=0,from_=0,to=100,
                       tickinterval 25,
                       command=self.updateCercle)
self.scaleCercle.set(50)
```

Application : initialisation

MainWindow : __init__(self)

```
self.bw=self.canvas.create_window(self.x,self.y,
                                  window=self.buttonStart)
self.xscale,self.yscale=200,200
self.sw=self.canvas.create_window(self.xscale,
                                   self.yscale,
                                  window=self.scaleCercle)
self.xcercle,self.ycercle=100,100
self.cercle=self.canvas.create_oval(self.x, self.y,
                                    self.x+self.xcercle,
                                    self.y+self.xcercle,
                                    fill='yellow',
                                    outline='black')
```

Application : interaction

MainWindow: méthodes d'interaction

```
def stop(self):
    self.run=0
    self.buttonStart.configure(text="Start",
                               command =self.start)

def start(self):
    self.buttonStart.configure(text="Stop",
                               command=self.stop)
    self.run=1
    self.animation()
```

Application : animation

MainWindow: appel récursif de méthode

```
def animation(self) :
    if self.run ==0:
        return
    self.x += randrange(-60, 61)
    self.y += randrange(-60, 61)
    self.canvas.coords(self.cercle,
                       self.x, self.y,
                       self.x+self.xcercle,
                       self.y+self.ycercle)
    self.libelle.config(text='Cherchez en \
                              %s %s' \
                              % (self.x, self.y))
    self.after(250,self.animation)
```

Application : interaction

MainWindow: modification d'affichage

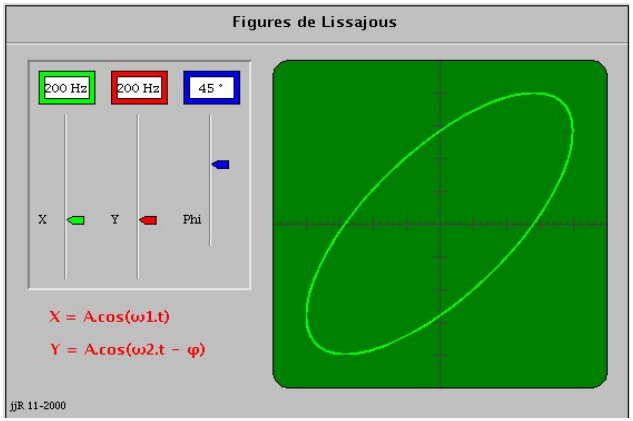
```
def effacer(self):
    self.canvas.delete(self.cercle)
def afficher(self):
    self.cercle=self.canvas.create_oval(self.x,self.y,
                                        self.x+self.xcercle,
                                        self.y+self.ycercle,
                                        fill='yellow')
def updateCercle(self,x):
    self.canvas.delete(self.cercle)
    self.xcercle, self.ycercle=int(x),int(x)
    self.cercle=self.canvas.create_oval(self.x,self.y,
                                        self.x+self.xcercle,
                                        self.y+self.ycercle,
                                        fill='yellow')
```

Oscilloscope

Projet de Labo

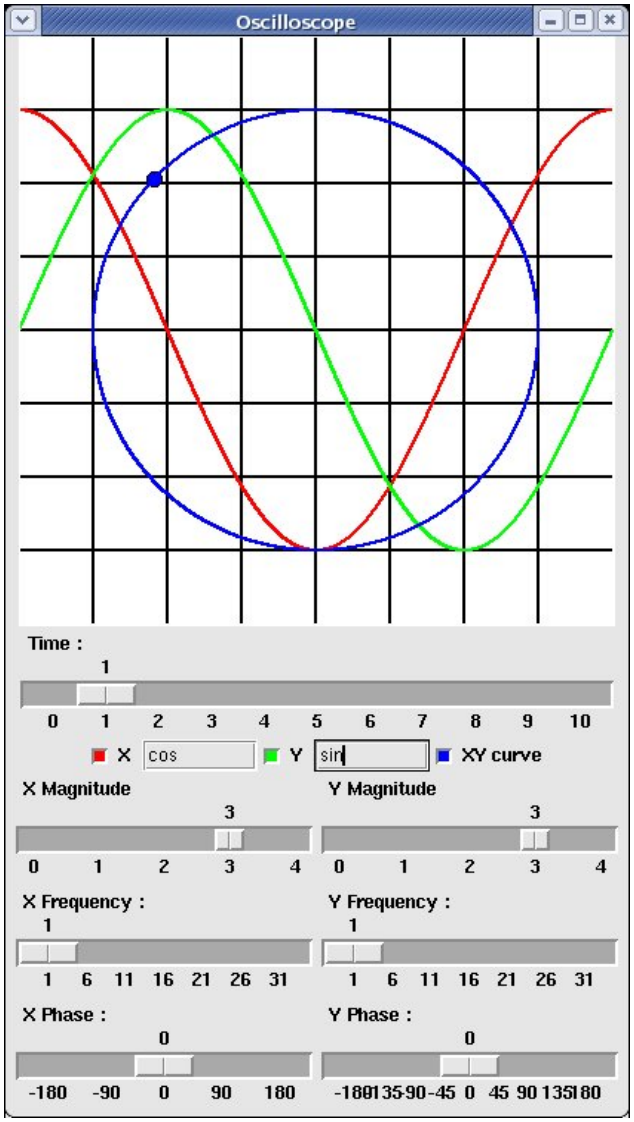
- visualisation de mouvement vibratoire harmonique
- contrôle en Amplitude, Fréquence et Phase
- gestion de la base de temps
- oscilloscope en mode XY

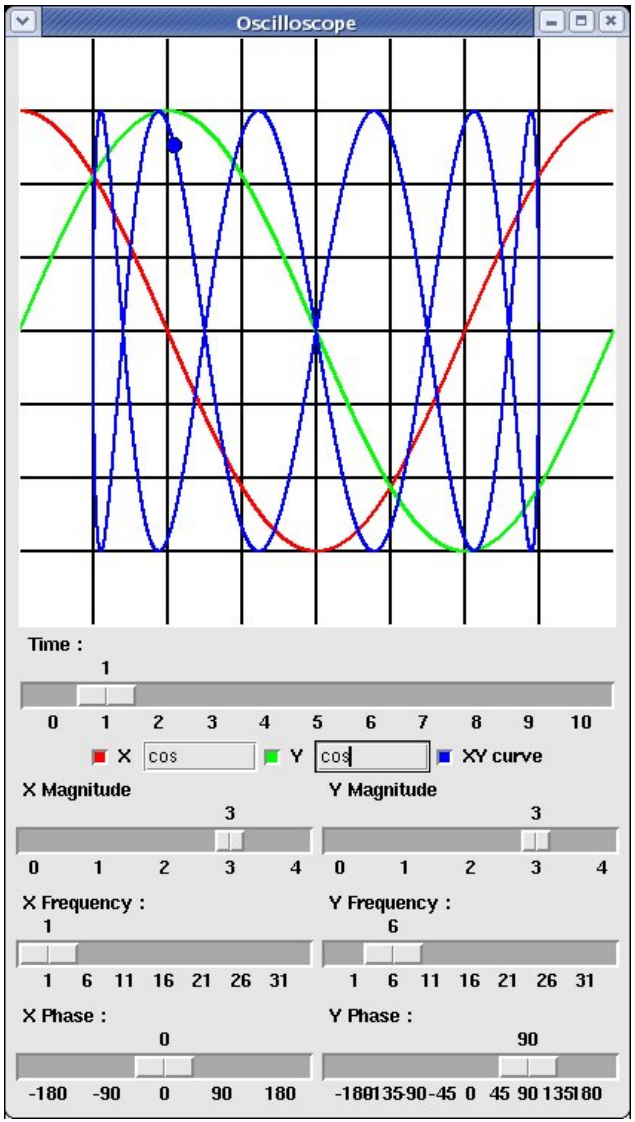
Exemple :



Oscilloscope

Autres exemples :

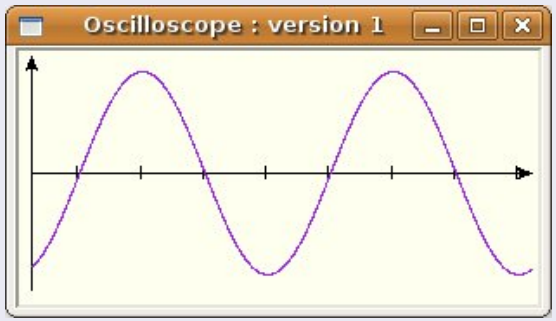




Mouvement vibratoire harmonique

Définition : $e = A\ sin(2\pi\omega t + \phi)$

- e : élongation
- A : amplitude
- ω : pulsation, fréquence
- t : temps
- ϕ : phase



OscilloGraphe : version 1

OscilloGraphe : initialisation

```
class OscilloGraphe(Canvas):
    def __init__(self,boss=None,larg=200,haut=150):
        Canvas.__init__(self)
        boss.title("OscilloGraphe : version 1")
        self.configure(width=larg,height=haut)
        self.larg,self.haut=larg,haut
        self.courbe=[]
```

OscilloGraphe : version 1

OscilloGraphe : repère d'affichage

```
    def repere(self, steps):
        "Repere d'affichage"
        self.create_line(10,self.hauteur/2,
                         self.largeur,self.hauteur/2,
                         arrow=LAST)
        self.create_line(10,self.hauteur-5,
                         10,5,
                         arrow=LAST)
        pas=(self.largeur-10)/steps*1.
        for t in range(1,steps+2):
            stx =t*pas
            self.create_line(stx,self.hauteur/2-4,
                             stx,self.hauteur/2+4)
        return
```

OscilloGraphe : version 1

OscilloGraphe : calcul du mouvement

```
def calculVibration(self,
                    frequence=1,phase=0,amplitude=10):
    "calcul de l'elongation sur 1 seconde"
    del self.courbe[0:]
    pas=(self.largeur-10)/1000.
    for t in range(0,1001,5):
        e=amplitude*sin(2*pi*frequence*t/1000-phase)
        x=10+t*pas
        y=self.hauteur/2-e*self.hauteur/25
        self.courbe.append((x,y))
    return
```

OscilloGraphe : version 1

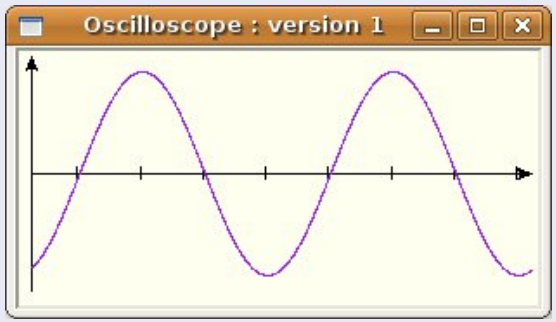
OscilloGraphe : Affichage de la courbe

```
def traceCourbe(self,couleur='red'):
    "visualisation de la courbe"
    if len(self.courbe) > 1 :
        n = self.create_line(self.courbe,
                             fill=couleur,
                             smooth=1)
    return n
```

OscilloGraphe : version 1

Application de test

```
if __name__ == '__main__':
    root = Tk()
    oscillo= OscilloGraphe(root,300)
    oscillo.configure(bg='ivory',bd=2,relief=SUNKEN)
    oscillo.repere(8)
    oscillo.calculVibration(2,1.2,10)
    oscillo.traceCourbe()
    oscillo.pack()
    root.mainloop()
```



Classe OscilloGraphe : version 1

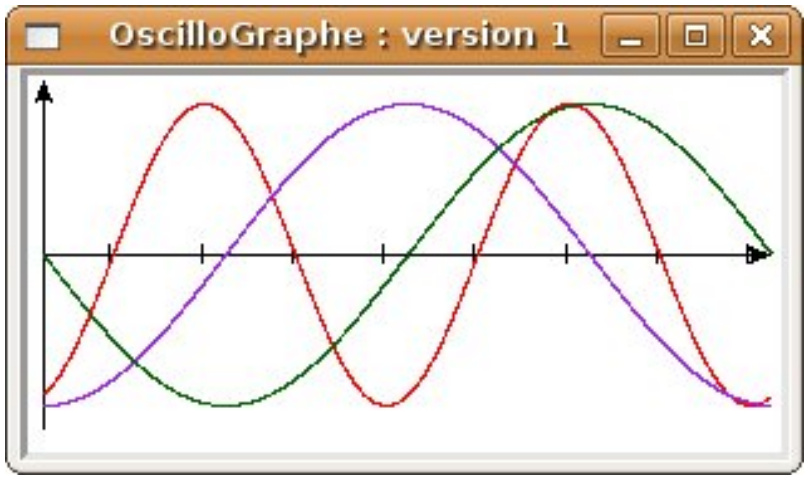
Utilisation du Module `oscillo`

```
from oscillo import *
root = Tk()
oscillo = OscilloGraphe(root,300)
oscillo.configure(bg='white', bd=3, relief=SUNKEN)
oscillo.repere(8)
oscillo.calculVibration(2, 1.2, 10)
oscillo.traceCourbe()
...
```

Classe OscilloGraphe : version 1

Utilisation du Module `oscillo`

```
...
oscillo.calculVibration(phase=1.57)
oscillo.traceCourbe(couleur='purple')
oscillo.calculVibration(phase=3.14)
oscillo.traceCourbe(couleur='dark green')
oscillo.pack()
root.mainloop()
```



Python : Programmation procédurale

Initialisation : variables, fonctions

```
# Importation de variables,fonctions, modules externes
import sys
from math import sqrt, sin, acos
# Variables, fonctions necessaires au programme
def spherical(x,y,z):
    r, theta, phi = 0.0, 0.0, 0.0
    r = sqrt(x*x + y*y + z*z)
    theta = acos(z/r)
    if theta == 0.0:
        phi = 0.0
    else :
        phi = acos(x/(r*sin(theta)))
    return r, theta, phi
```

Python : Programmation procédurale

Traitements de données, sortie de programme

```
# Traitements
x = input('Entrez la valeur de x : ')
y = input('Entrez la valeur de y : ')
z = input('Entrez la valeur de z : ')
print "Les coordonnees spheriques du point :", x,y,z
print "sont : ", spherical(x,y,z)
# sortie de programme
sys.exit(0)
```

Python : Programmation Orientée Objet

définition d'une classe

```
class Point:
    """point 2D"""
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __repr__(self):
        return "<Point('%s','%s')>" \
            % (self.x, self.y)
```

Python : Programmation Orientée Objet

Composition

```
class Rectangle:
    """Un rectangle A UN coin superieur gauche"""
    def __init__(self, coin, largeur, hauteur):
        self.coin = coin
        self.largeur = largeur
        self.hauteur = hauteur
    def __repr__(self):
        return "<Rectangle('%s','%s','%s')>" \
            % (self.coin,self.largeur, self.hauteur)
```

Python : Programmation Orientée Objet

Heritage

```
class Carre(Rectangle):
    """Un carre EST UN rectangle particulier"""
    def __init__(self, coin, cote):
        Rectangle.__init__(self, coin, cote, cote)
        self.cote = cote
    def __repr__(self):
        return "<Carre('%s','%s')>" \
            % (self.coin,self.cote)
```

Python : Programmation Orientée Objet

Application de test

```
if __name__ == '__main__':
    p=Point(10,10)
    print p
    print Rectangle(p,100,200)
    print Carre(p,100)
```

Test

```
{logname@hostname} python classes.py
<Point('10','10')>
<Rectangle('<Point('10','10')>','100','200')>
<Carre('<Point('10','10')>','100')>
```

Bibliographie

Documents

- Gérard Swinnen :
  “Apprendre à programmer avec Python” (2005)
  “Apprendre à programmer avec Python 3” (2010)
- Guido van Rossum:
  “Tutoriel Python Release 2.4.1” (2005)
- Fredrick Lundh :
  “Plongez au coeur de Python” (2006)
- Mark Pilgrim :
  “An introduction to Tkinter” (1999)
- John W. Shipman :
  “Tkinter reference: a GUI for Python” (2006)
- John E. Grayson :
  “Python and Tkinter Programming” (2000)

Bibliographie

Adresses “au Net”

- inforef.be/swi/python.htm
- python.developpez.com
- www.limsi.fr/Individu/pointal/python.html
- wiki.python.org/moin/TkInter
- www.jchr.be/python/tkinter.htm
- effbot.org/tkinterbook
- www.jchr.be/python/tkinter.htm
- www.pythonware.com/library/tkinter/introduction