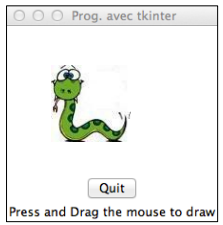
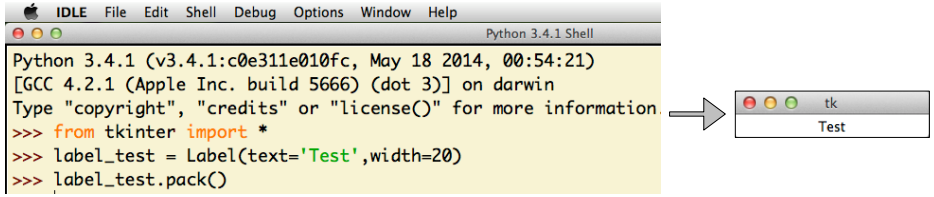
Programmation d'une interface graphique

(tkinter)



Introduction

- **tkinter** : bibliothèque provenant de l'extension graphique **Tk** du langage **Tcl** (1988).
- L'extension a largement dépassé le cadre de Tcl/Tk.
- Tk utilisable dans différents langages (Perl, Python, Ruby, ...)
- En python, c'est **tkinter** : *Tk interface*
- Permet de réaliser un **GUI** (*Graphical User Interface*)
- GUI : une interface homme-machine permettant à l'utilisateur d'interagir avec la machine autrement que par le toplevel.



Exemples sur http://fsincere.free.fr/isn/python/cours_python_tkinter.php

Schéma de la programmation classique

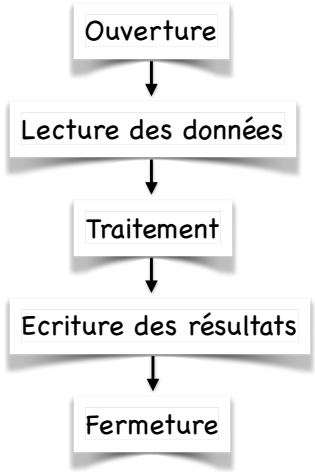
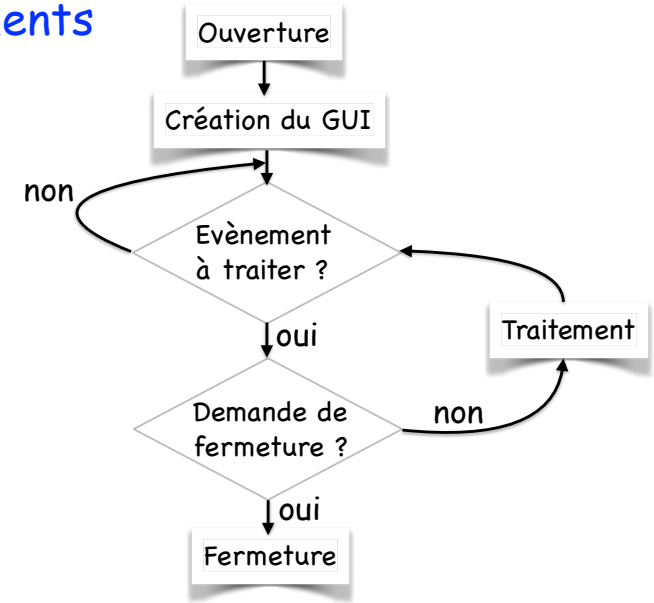


Schéma de la programmation dirigée par les évènements

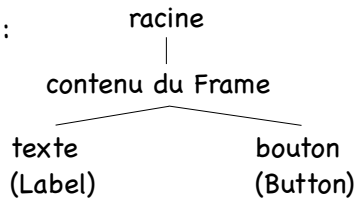


Les widgets : gadgets de fenêtrage

• **widget** = window gadget : tous les composants de l'écran (fenêtres, boutons, ascenseurs, ...)

• Tk assure la gestion des widgets :
  - fenêtre de composants (Frame)
  - fenêtre de dessin (Canvas)
  - affichage d'informations (Label, Message, Text)
  - interaction avec les composants (Button, Scale, ...)
  - saisie de texte (Entry)

• Les widgets sont éléments d'une hiérarchie :



Structure du programme


```
from tkinter import *                                   initialisation

root=Tk()                                               création de la fenêtre maître

message = Label(root, text = 'Un premier exemple !',\
                fg = 'red')                             définition des composants

button = Button(root, text = 'Quitter',command = root.destroy)

message.pack()                                          affichage des composants
button.pack()

root.mainloop()                                         lancer le gestionnaire d'événements
                                                        dans la fenêtre root
```

N.B. Pour quitter cette mini-application et revenir au toplevel : `root.destroy()` ou bien cliquer dans la case de fermeture de la fenêtre !

Création de widgets

• Chaque widget est un objet python.

• Création de widget : passer le parent de la hiérarchie en premier argument de la fonction de création.

• Les widgets ont beaucoup d'options de configuration (couleur, position, etc...).

```
from tkinter import *
root=Tk()
label_etiquette=Label(root,text='Exemple')
label_etiquette.pack()
```

parent — option de configuration

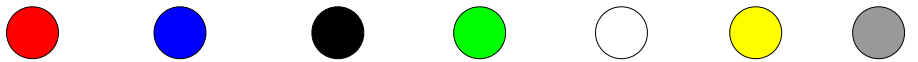
```
label_etiquette=Label(root,text='Un premier exemple !',fg='red')
button_quitter=Button(root,text='Quitter',command=root.destroy)
```

création bouton — interaction avec composant

Les couleurs

• Une couleur peut s'exprimer :

  - soit par une chaîne de caractères constante :



  - soit par un mélange RGB à la place du texte, en hexadécimal :

```
label_etiquette['fg']='#xxyyzz'
```

Ex : la couleur jaune s'obtient avec un mélange de rouge et de vert:

```
jaune = '#ffff00'
label_etiquette['fg'] = jaune
```

http://www.w3schools.com/html/html_colors.asp

Exemple : intégration à la Monte-Carlo

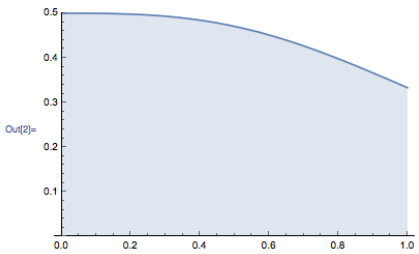
• Il s'agit de calculer l'intégrale définie d'une fonction dont une primitive est difficile ou impossible à calculer. **En utilisant des probabilités**.

• Soit à calculer une approximation de l'intégrale sur [0,1] :

$$\int_0^1 \frac{1}{x^3+2}\,dx$$

• Voici la courbe de la fonction sur [0,1] obtenue avec Mathematica :

```
In[1]:= f[x_] = 1 / (2 + x^3);
In[2]:= Plot[f[x], {x, 0, 1}, Filling -> Bottom, AxesStyle → Black, PlotRange → {0, 0.5}]
```



• Graphiquement, l'intégrale est un peu inférieure à 0.5. D'ailleurs le logiciel Mathematica donne comme approximation numérique :

```
In[33]:= NIntegrate[f[x], {x, 0, 1}]
Out[33]= 0.450822
```

• Mathematica utilise des techniques sophistiquées. Nous nous proposons d'effectuer ce calcul approché avec une **méthode de Monte-Carlo**. Nous allons jeter un grand nombre N de points au hasard dans le rectangle [0, 1] x [0, 0.5] contenant la courbe, et noterons le nombre S de ceux qui tombent dans la surface entre la courbe et Ox.

• La **probabilité** p qu'un point tombe dans cette surface peut se calculer sous la forme de deux proportions distinctes :

-soit sous la forme p = S / N
-soit sous la forme p = aire(courbe) / aire(rectangle) = I / 0.5

• On en déduit la valeur de l'intégrale : $I \approx 0.5\frac{S}{N}$
et tout revient à calculer S.

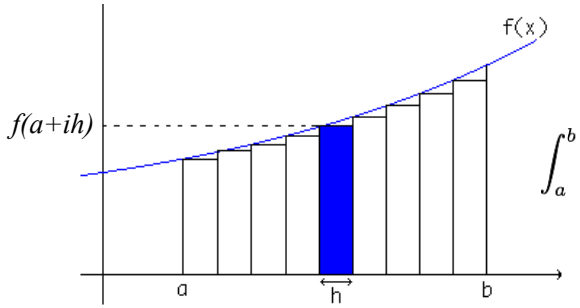
Implémentation pure de la stratégie de Monte-Carlo

• Sans interface graphique, **juste le calcul mathématique**. Nous travaillons dans le rectangle [0, 1] x [0, 0.5].

```
from random import random     # pour tirer dans [0,1[

def f(x) :
    return 1 / (2 + x**3)

def monte_carlo(N) :          # avec N points aléatoires
    succès = 0                # le nombre de succès
    for i in range(N) :
        (x,y) = (random(), random() * 0.5)
        if y < f(x) : succès = succès + 1
    return 0.5 * succès / N
```

```
>>> monte_carlo(50000)        # 50000 points au hasard
0.45063                       # approximation de l'intégrale
```


• Pour simple rappel, calcul d'une intégrale avec la **méthode de Riemann**. On approche la fonction continue f à intégrer par une **fonction en escalier** en découpant l'intervalle d'intégration en N points équirépartis.



$$\int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f(a+ih)$$

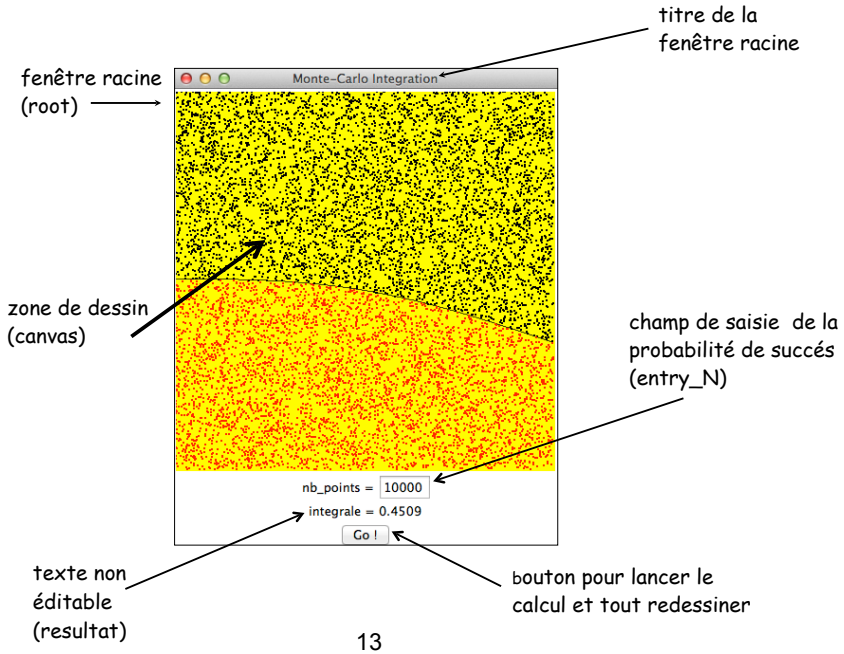
formule de Riemann

```
def integrale_riemann(f,a,b,N) :
    h = (b - a) / N
    return h * sum(f(a + i*h) for i in range(N))
```

```
>>> integrale_riemann(f,0,1,1000)
0.45090543481930634
```

Simplicité de l'écriture en Python !...

L'interface graphique avec tkinter



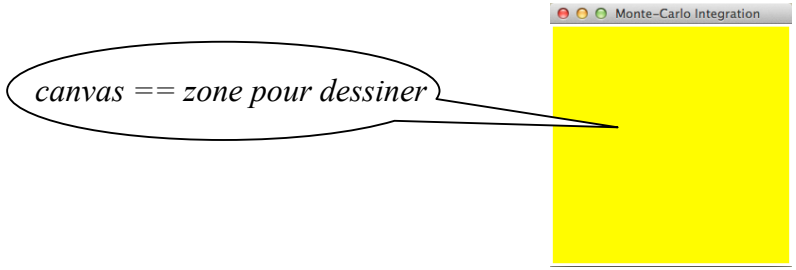
Création du canvas 400 x 400

```
from tkinter import *
```
pour la création de l'interface

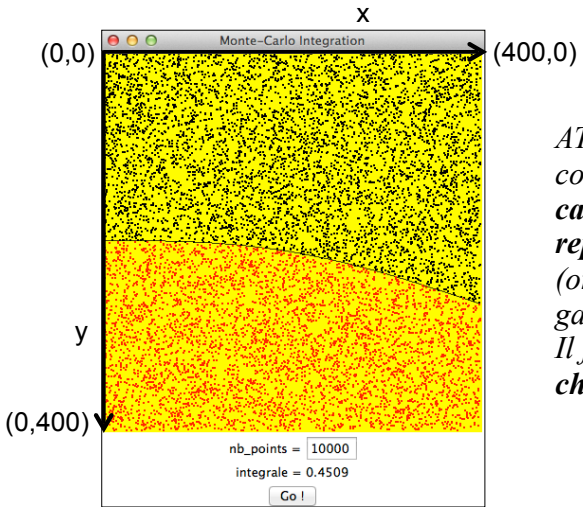
```
root=Tk()
root.title('Monte-Carlo Intégration')
```
création de la fenêtre maître et initialisation du titre

```
canvas = Canvas(root,width=400,height=400,bg='yellow')
```
définition d'un composant

```
canvas.pack()
```
affichage du composant



Le canvas : une zone à dessiner, son système de coordonnées

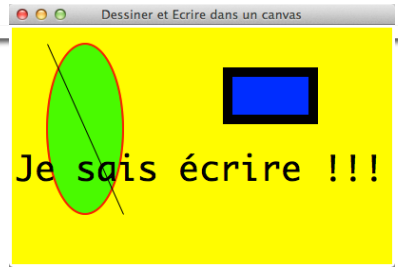


ATTENTION : ne pas confondre le ***repère du canvas*** *(ci-contre) et le* ***repère mathématique*** *(origine en bas à gauche, Oy vers le haut). Il faudra faire un* ***changement de repère*** *!*

• Mise à part la tortue, il est usuel en programmation de prendre l'**origine des coordonnées en haut à gauche**, l'axe **Oy vers le bas** !
Il reste à savoir dessiner dans un canvas…

Intermède - Comment dessiner dans un canvas ?

```
from tkinter import *

root = Tk()

root.title('Dessiner et Ecrire dans un canvas')
canvas = Canvas(root,width=400,height=300,bg='yellow')
canvas.pack()

canvas.create_oval(40,20,120,200 ,outline='red',fill='green', width=2)
canvas.create_rectangle(230,50,320,100,fill='blue',width=10)
canvas.create_line(40,20,120,200)
canvas.create_text(200,150,text='Je sais écrire !!!',font=('monaco',36))

root.mainloop()
```



Widgets : Entry pour saisir une donnée et Label pour afficher un texte

• Le composant dans lequel l'utilisateur peut entrer le nombre de points N est dans la classe **Entry** de `tkinter`. Plaçons ce composant dans le canvas, précédé d'un composant **Label** pour un texte non éditable. Nous incorporons ces deux composants dans un **panneau horizontal invisible** `hpanel` de type **Frame** qui sera ajouté à la verticale, centré par défaut.

```
hpanel = Frame(root)
label_N = Label(hpanel,text='nb_points =')
label_N.pack(side=LEFT)
entry_N = Entry(hpanel,width=5)
entry_N.pack(side=LEFT)
entry_N.insert(0,'500')     # valeur initiale
hpanel.pack()
```

nb_points = 500

• *En réalité, Pack est un "gestionnaire de disposition" (il empile les composants à la verticale ou à l'horizontale). Il existe d'autres gestionnaires comme Grid (dans une grille) ou Place (n'importe où)…*

Récupération du contenu d'une Entry

• Au moment de lancer le calcul, on pourra récupérer le contenu de entry_N par `entry_N.get()` qui retournera une chaîne de caractères (qu'il faudra convertir en `int` !).

Association d'un objet StringVar au contenu d'un Label

• Lorsque le contenu d'un `Label` n'est pas fixe mais doit être mis à jour, il est usuel de l'associer à un objet `res` de la classe `StringVar`, doté d'une méthode `res.set(…)` immédiatement répercutée sur l'affichage !

```
res = StringVar()
label_res = Label(root,textvariable = res)
label_res.pack()
res.set('integrale = ?')
```

• Et à la fin du calcul :

```
res.set('intégrale = {}'.format(res))
```


Et enfin les deux boutons…

• Un **bouton** est un composant de la classe `Button` de `tkinter`. Son attribut le plus important est `command`, contenant une fonction d'arité 0. Cette fonction (le ***callback*** du bouton) sera automatiquement exécutée suite à un clic de l'utilisateur dans le bouton.

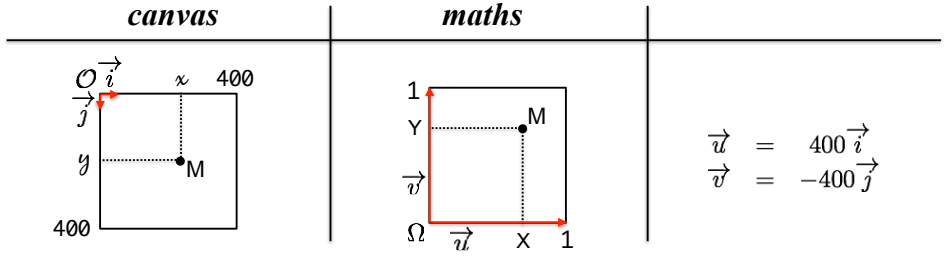
```
def dessiner() :                        # le callback de button
    pass                                # pas urgent

button = Button(root,text='Go !',command=dessiner)
button.pack()
```

• La fonction `dessiner()` sera chargée de tracer la courbe, de jeter des points au hasard dans le canvas, et d'en déduire l'intégrale approchée. Son code sera détaillé page 21… Pour l'instant, le bouton est actif mais ne réagit pas.

• Il faudra peut-être rajouter un bouton ***Quitter*** dont la commande est prédéfinie et se nomme `root.destroy`.

Sur le changement de repère…

• Nous avions dit à la page 15 qu'il y avait deux repères en jeu. L'un d'eux $(\mathcal{O}, \vec{i}, \vec{j})$ est associé au canvas (unités en pixels). L'autre $(\Omega, \vec{u}, \vec{v})$ est le repère mathématique (unité = 400 pixels).



• En projetant la relation $\overrightarrow{OM} = \overrightarrow{O\Omega} + \overrightarrow{\Omega M}$ sur les deux axes, on obtient les équations de **changement de repère** :

$$\begin{cases} x = 400X \\ y = 400(1-Y) \end{cases}$$

Dessin de la courbe de f

• Procédure de tracé du segment M_1M_2 où M_1 et M_2 sont donnés en unités mathématiques :

```
def segment(X1,Y1,X2,Y2) :        # en unités mathématiques
    # d'abord le changement de repère
    x1 = 400 * X1 ; y1 = 400 * (1 - Y1)
    x2 = 400 * X2 ; y2 = 400 * (1 - Y2)
    canvas.create_line(x1,y1,x2,y2)              # le tracé
```

• Tracé de la courbe de la fonction f en unités mathématiques :

```
def plot() :    # 1 unité mathématique sur Ox = 400 pixels
    (X,Y) = (0,f(0))         # en unités mathématiques
    dX = 1/400
    for i in range(0,400) :  # pour chaque pixel sur Ox
        Xs = X + dX
        Ys = f(Xs)
        segment(X,Y,Xs,Ys)
        (X,Y) = (Xs,Ys)
```


• Et enfin le ***callback*** du bouton : la fonction `dessiner` qui organise l'expérience de Monte-Carlo.

```
def dessiner() :
    canvas.delete(ALL)
    plot()
    N = int(entry_N.get())
    succès = 0
    for i in range(N) :
        (X,Y) = (random(),random())     # coordonnées mathématiques
        (x,y) = (X*400,400*(1-Y))       # coordonnées du canvas
        if Y < f(X) :                   # dans la surface cherchée ?
            succès = succès + 1
            canvas.create_oval(x,y,x,y,outline='red')
        else :
            canvas.create_oval(x,y,x,y,outline='black')
    proba = succès / N
    res.set('integrale = {}'.format(proba))
```


OPTIONNEL$_1$: comment programmer une **ANIMATION** ?

• Thème approfondi au semestre 2 en langage Scheme. Pour faire simple : la philosophie est **MVC** (Modèle-Vue-Contrôleur).

• Concept de base : **le monde** == ensemble des variables qui gouvernent le phénomène physique. Il s'agit du **modèle mathématique**. Aucun dessin. On commence par initialiser les variables du monde.

• Une fonction `dessiner()` sera chargée de dessiner une image d'après les variables du monde. Une seule image, et elle ne modifie pas le monde.

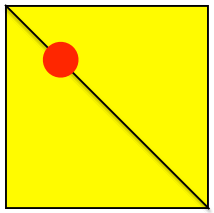
• Une fonction `suivant()` sera chargée de mettre à jour les variables du monde en prévision de la prochaîne image. Elle calcule ! Aucun dessin.

• Une fonction `final()` sera chargée de rendre `True` si le monde est dans état final, où l'animation doit stopper. Elle rend `False` sinon.

```
def animation() :                    # moteur d'une animation
    if not final() :
        dessiner()
        suivant()
        canvas.after(10,animation)   # horloge à 10 ms
```


OPTIONNEL$_2$: un exemple d'animation en style **MVC**

```
# anim-mvc.py
# Python en L1-Sciences, Nice, 2016-2017
# Utilisation de tkinter pour une animation simple dans le style "MVC"
# https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur
# Une balle qui descend le long de la diagonale du canvas et stoppe en bas

from tkinter import *

root = Tk()
root.resizable(False,False)
HAUTEUR = 400
LARGEUR = 400
canvas = Canvas(root, width=LARGEUR, height=HAUTEUR, bg='yellow')
canvas.pack()

# les constantes ne font pas partie du monde !
r = 20    # rayon de la balle
dx = 2    # vitesse horizontale de la balle
```



```
# LE MONDE EST CONSTITUE DE LA SEULE VARIABLE GLOBALE x
x  = 0    # abscisse de la balle (la trajectoire imposée rend y inutile)

def dessiner() :             # dessiner la Vue à partir du monde
    canvas.delete(ALL)
    canvas.create_line(0,0,LARGEUR,HAUTEUR)
    canvas.create_oval(x-r,x-r,x+r,x+r,fill='red')

def suivant() :              # mise à jour du Modèle (le monde)
    global x
    x = x + dx               # on avance du vecteur vitesse à chaque top
d'horloge !

def final() :                # le monde est-il dans un état final ?
    return x >= LARGEUR

def animation() :            # le Contrôleur de l'animation
    if not final() :
        dessiner()
        suivant()
        canvas.after(10,animation)     # timer de l'animation : 10 ms
    else :
        print('fini !')

animation()
```