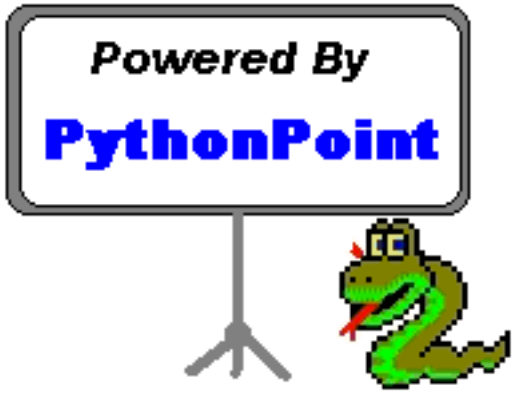
GUI programmation en Tcl, Perl et Python

par

Laurent Pierron

Mardi 30 mars 2002



Bienvenue à la GUI* programmation

...un cours pour vous apprendre à développer des *cliquodromes*.

GUI Programmation va vous initier à la programmation rapide d'interface utilisateur avec notamment la puissante bibliothèque d'objets graphiques Tk.

J'espère que ce cours vous donnera envie de développer vos propres interfaces utilisateurs, voir vos propres applications. Vous le verrez : c'est facile, c'est pas cher (logiciels gartuits) et ça peut rapporter gros (certains logiciels sont devenus des produits commerciaux).

(*) Graphical User Interface

Programmation Tcl/Tk

Tcl est un langage de commande basé sur l'évaluation de chaînes de caractères suivant le même principe que les langages de commande sous Unix (sh, csh, bash).

Un programme Tcl est une chaîne de caractères.

Le programme est divisé en lignes.

Une ligne est divisée en mots séparés par des espaces

Le premier mot d'une ligne est une commande et les autres mots de la ligne sont les paramètres de la commande.

Par des indcations syntaxiques les mots de la ligne peuvent contenir des variables ou des commandes à évaluer avant d'exécuter la commande.

Exemple basique de programme Tcl

Voici un programme **Tcl** de deux lignes :

```
set a 44
expr $a * 4
```

La première ligne a trois mots la commande est **set** avec deux paramètres *a* et *44*. L'effet de cette commande est d'associer au premier paramètre le second (effectue une sorte d'affectation).

La seconde ligne fait 4 mots, la commande qui y est associée place tous les paramètres dans une seule chaîne et évalue la chaîne comme étant une expression mathématique.

Dans les paramètres il y a une chaîne spéciale *$a* qui est remplacée par la valeur associée à *a*.

Le résultat de la seconde ligne renvoie la chaîne *166*. Pour en savoir plus sur **Tcl**, se reporter à mon cours de décembre 2001.

Convertisseur Franc-Euro

Ce premier programme **Tcl/Tk** doit permettre de saisir une valeur en francs et afficher la contre-valeur en euros. Voici l'interface graphique que l'on souhaite réaliser :

Code Tcl/Tk pour le convertisseur.

```
entry .franc -textvariable
franc button .conv -text " conversion " \
       -command {set euro [expr $franc / 6.55957]}
label .euro -textvariable euro
label .titre -text "Conversion francs-euros\n saisissez la valeur en fran
pack .titre .franc .conv .euro
```

- Crée une zone de saisie
- Crée un bouton de conversion
- Crée une zone d'affichage
- Place un titre
- Affiche tout à l'écran

Saisie de texte (1)

```
entry .franc -textvariable franc
```

La commande **entry** est une commande de la bibliothèque Tk. Cette commande crée une zone pour la saisie d'une chaîne de caractères d'une ligne.

Le premier paramètre *.franc* est le nom de l'objet graphique qui sera créé puis utilisé comme une commande dans le reste du programme. Ce nom doit toujours commencer par un point '**.**', nous expliquerons plus tard pourquoi.

Toutes les créations d'objets graphiques *widget* auront comme premier argument le nom que l'on souhaite donner à l'objet.

Saisie de texte (2) : paramètres

```
entry .franc -textvariable franc
```

Les autres paramètres sont des couples (*nom de paramètre*, *valeur du paramètre*). Les noms de paramètres commencent par un tiret '**-**'. Pour connaître la liste des paramètres de chaque widget on se reportera à la documentation du widget.

Le paramètre *-textvariable* permet d'associer une variable à une zone de saisie, quand la zone de saisie est modifiée la variable est modifiée. Quand la variable est modifiée la zone de saisie est mise à jour.

Exercice : trouvez la documentation dans votre environnement de développement Tcl. **tclhelp** ou pages de manuel sous Unix. Pages HTML sur Mac. Aide de Windows sous Windows. Sur le web : *http://www.scriptics.com/man/tcl8.3/*

Widget Button

```
button .conv -text " conversion " \
             -command {set euro [expr $franc / 6.55957]}
```

La commande **button** crée un widget, qui réagit à des évènements de la souris.

Appui sur le bouton de la souris lance le programme associé au paramètre -*command*. Ici le programme est une chaîne **Tcl**, qui place dans la variable *euro* la valeur de la variable *franc* divisée par *6,55957*.

-*text* décrit l'étiquette à placer sur le bouton.

Affichage de texte court

```
label .euro -textvariable euro
label .titre \
      -text "Conversion francs-euros\n saisissez la valeur en francs."
```

La commande **label** crée un widget contenant un texte court éventuellement sur plusieurs lignes.

Ici création de deux widgets, un contient un texte défni par *-text*, l'autre est associé à une variable *-textvariable*. Toute modification de la variable est automatiquement répercutée sur l'affichage du texte.

Comme pour tous les widgets, la taille du widget **label** est automatiquement calculée en fonction du contenu de ce widget. **Magique, non !!**

Organisation spatiale et affichage des widgets

```
pack .titre .franc .conv .euro
```

Les commandes précédentes créent les objets graphiques (*widgets*) en mémoire, mais ne les affichent pas. Pour les afficher, il faut utiliser un des trois gestionnaires de géométrie de **Tcl**.

La commande **pack** réalise le positionnement des *widgets* dans la fenêtre de l'application.

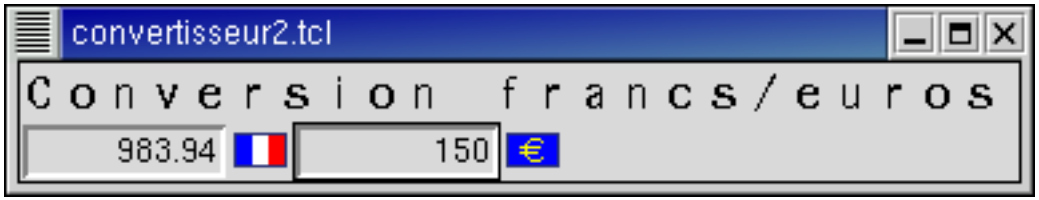
Le *Packer* est le gestionnaire le plus simple, il positionne les *widgets* relativement les uns par rapport aux autres. Par défaut, les *widgets* sont empilés les uns au-dessus des autres.

Exercice : tapez ce programme en interactif sous le *shell* **wish**.

Convertisseur Franc-Euro (II)

Ce second programme doit permettre une conversion bi-directionnelle, automatique à la volée. Il doit effectuer les arrondis et il doit être robuste aux erreurs de saisie.

Voici l'interface graphique que l'on souhaite réaliser :



Code Tcl/Tk pour le convertisseur

```
set images [file dirname $argv0]/images

label .titre -text "Conversion francs/euros" -font "bold"
pack .titre

label .lfr -image [image create photo -file $images/frf.gif]
entry .franc -width 10 -textvariable franc -justify right
pack .franc .lfr  -side left

label .leu -image [image create photo -file $images/eur.gif]
entry .euro -width 10 -textvariable euro -justify right
pack .euro .leu  -side left

bind .franc <KeyRelease> {set euro  [format %%.2f \
                        [expr ([string map {, .} $franc]) / 6.55957]]}
bind .euro <KeyRelease>  {set franc [format %%.2f \
                         [expr ([string map {, .} $euro]) * 6.55957]]}

proc bgerror {e} {
    bell
    foreach i {.franc .euro} {
        $i delete 0 end    ;# Vide le champ
        $i insert 0 0.00   ;# Met 0.00 dans le champ
        $i icursor 1       ;# Positionne le curseur avant l '.'
}    }
```

Style des textes

```
label .titre -text "Conversion francs/euros" -font "bold"
pack .titre
```

Le paramètre *-font* permet d'associer un style de caractère au widget. Le style de caractère peut être exprimé sous la forme d'un nom de police X11 ou par un texte exprimant une police et ses attributs, par exemple *"Adobe Times 12 italic"*. Dans l'exemple *Tk* utilisera la police par défaut en gras (*bold*).

Ici le widget *.titre* est tout de suite mis en place avec la commande **pack**.

Gestion des images

```
label .lfr -image [image create photo -file $images/frf.gif]
entry .franc -width 10 -textvariable franc -justify right
pack .franc .lfr -side left
```

La commande **image** permet de créer un objet image à partir d'un fichier. En standard seules les images de type PPM et GIF sont reconnues. L'image peut ensuite être utilisée dans différents widgets (label, button, etc.). Dans notre exemple elle servira à créer un widget **label** grâce à l'option *-image*.

Le widget **entry** a quelques paramètres supplémentaires, la taille de la zone de saisie (*-width*), l'alignement (*-justify*).

La zone de saisie et son étiquette sont immédiatement affichées, avec l'étiquette à gauche de la zone de saisie.

La même chose est faite pour la saisie des euros.

Gestion des évènements

```
bind .franc <KeyRelease> {
     set euro  [format %%.2f [expr ($franc) / 6.55957]]}
bind .euro <KeyRelease>  {
     set franc [format %%.2f [expr ($euro) *6.55957]]}
```

La commande **bind** permet d'associer une action à un évènement survenant sur un *widget*. En jargon on appelle cette action un *callback*.

Le premier paramètre de la commande est un *widget* ou une famille de *widget*, le second un évènement (ici une touche que l'on relâche) et le troisième la commande à appeler (ici un programme complet).

Le programme**Tcl** effectue la conversion, l'arrondit et la place dans la variable opposée. **Tk** met à jour en direct l'écran. Le passage d'une saisie euro vers franc se fait par clic dans la zone de saisie ou avec la tabulation. Tout celà est géré automatiquement par **Tk**.
Magique non !!

Gestion des erreurs

```
proc bgerror {err} {
    bell
    foreach i {.franc .euro} {
       $i delete 0 end  ;# Vide le champ
       $i insert 0 0.00 ;# Met 0.00 dans le champ
       $i icursor 1     ;# Positionne le curseur avant l '.'
} }
```

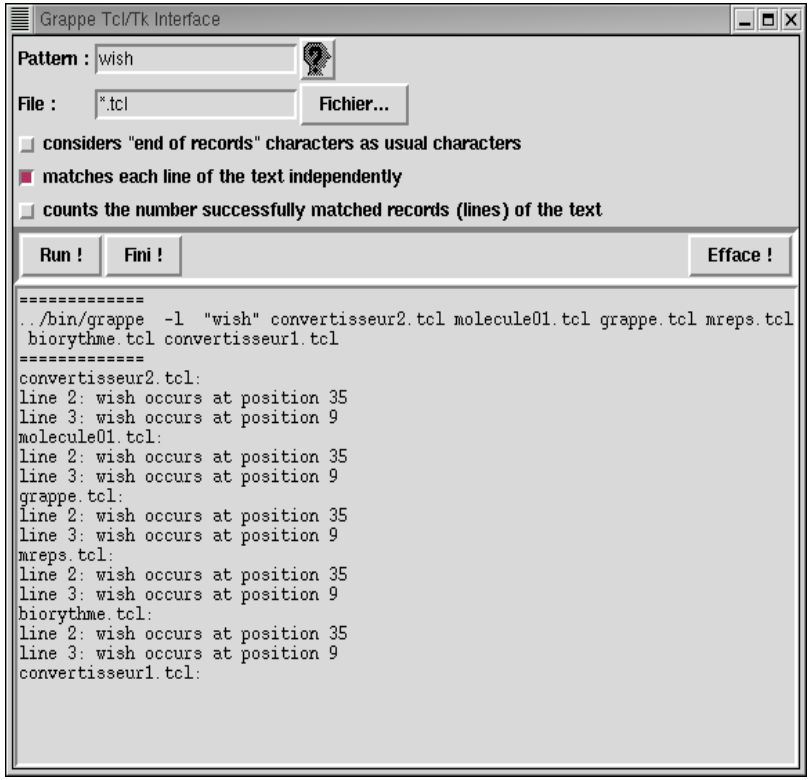
Quand une erreur survient dans **Tk**, la commande **bgerror** est appelée avec l'erreur en paramètre. Il suffit de réécrire cette commande **bgerror** pour traiter l'erreur.

Ici on fait sonner une cloche (**bell**) et on remet à zéro les champs, en utilisant des méthodes (*delete* et *insert*) appliquées aux objets graphiques. Chaque *widget* est en fait une nouvelle commande dont le premier argument est une méthode. Se reporter à la documentation pour connaître les méthodes des différents objets.

Exercice : tapez le programme et étudiez son comportement en tapant différents types de valeur dans les champs de saisie.

Interfaçage de programmes

Dans cette section, on va interfacer un programme existant sous forme de commandes Unix. L'objectif de cette interface est d'avoir une aide à la saisie des paramètres des programmes et un contrôle sur la visualisation des résultats.



Grappe : recherche de séquences

Grappe est un programme de recherche de sous-chaîne dans des fichiers. Il prend en paramètre une chaîne, un ou plusieurs fichiers et des drapeaux (options simples).

```
santifontaine tcl 72 % ../bin/grappe -l wish *.tcl
biorythme.tcl:
line 2: wish occurs at position 35
line 3: wish occurs at position 9
convertisseur1.tcl:
convertisseur2.tcl:
line 2: wish occurs at position 35
line 3: wish occurs at position 9
grappe.tcl:
line 2: wish occurs at position 35
line 3: wish occurs at position 9
molecule01.tcl:
line 2: wish occurs at position 35
line 3: wish occurs at position 9
```

Description de TkGrappe.

Ecran divisé en 4 parties :

- Partie saisie des paramètres : chaîne+fichier
- Partie des options
- Boutons de pilotage de l'application
- Texte résultat

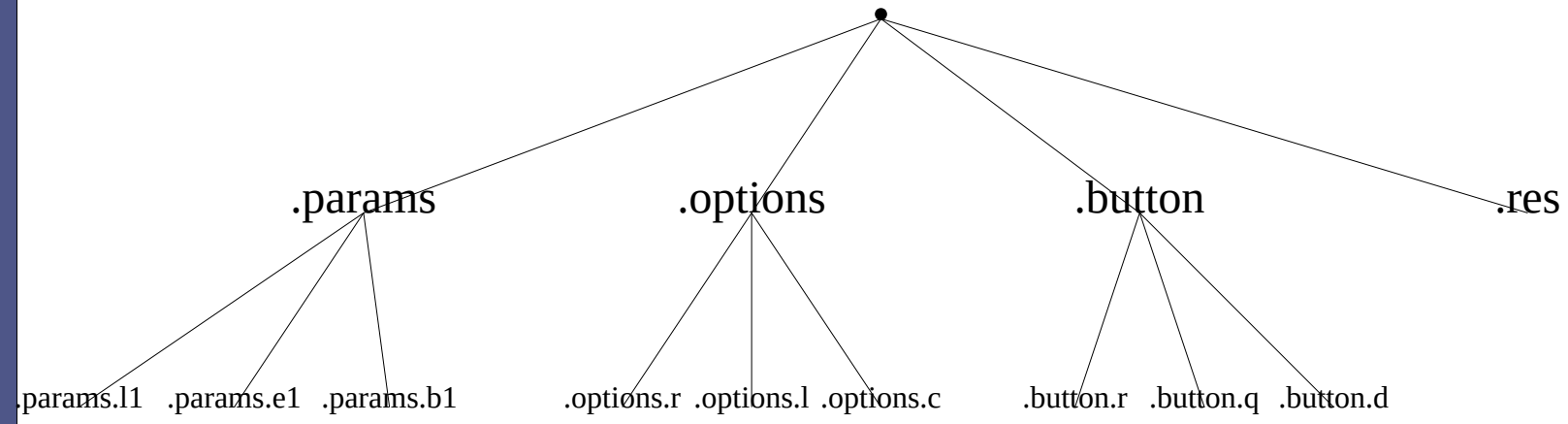
Exécution de programmes Unix

```
santifontaine tcl 74 % tclsh
%  exec ../bin/grappe -l wish mreps.tcl
mreps.tcl:
line 2: wish occurs at position 35
line 3: wish occurs at position 9
```

La commande **exec** passe à Unix sa liste d'arguments et retourne le résultat sous la forme d'une chaîne de caractères. On voit le résultat ci-dessus dans un shell **Tcl** interactif.

Organisation hiérarchique de TkGrappe

Les widgets sont organisés hiérarchiquement, suivant l'inclusion des widgets les uns dans les autres. La racine de la hiérarchie est **.**, d'où l'importance de commencer les noms par **.**. Les widgets *frame*, *toplevel*, *menu* et *menubar* peuvent contenir des autres widgets. Les différents niveaux de la hiérrachie sont séparés par des **.**. Chaque sous-niveau peut avoir un gestionnaire de géométrie différent.



Frame Widget

```
frame .params
label  .params.l1 -text "Pattern :"
entry  .params.e1 -textvariable pattern
button .params.b1 -text Help  -bitmap questhead \
  -command {exec mozilla http://www.loria.fr/~kucherov/software/grappe/ &
```

La commande **frame** crée un widget pouvant contenir d'autres widgets. Les widgets inclus dans le *frame* ont leur nom commençant par *nom_du_frame*..

Dans l'exemple ci-dessus destiné à la saisie de la chaîne à chercher, il y a trois widgets inclus dans *.params* : *.params.l1*, *.params.e1* et *.params.b1*.

button a un nouveau paramètre -*bitmap*, qui permet d'afficher une icône prédéfinie (le cas ici) ou venant d'un fichier. La *commande* associée à ce bouton permer d'afficher dans un navigateur Web la documentation du programme *Grappe*.

Lecture de nom de fichier

```
label .params.l2 -text "File :"
entry .params.e2 -textvariable file
button .params.b2 -text Fichier...
       -command {set file [tk_getOpenFile]}
```

Encore trois widgets dans le **frame** .*params* pour la saisie du nom de fichier.

La commande **tk_getOpenFile** ouvre un dialogue de saisie pour sélectionner un fichier. On remarquera l'imbrication des commandes.

Gestionnaire de géométrie : Grid

```
grid .params.l1 .params.e1 .params.b1 -sticky w
grid .params.l2 .params.e2 .params.b2 -row 1 -sticky w
pack .params -anchor w
```

La commande **grid** place les widgets dans une grille, comme sur un tableau.

6 widgets sont placés dans un tableau de 2 lignes sur 3 colonnes. La première case (ligne 0, colonne 0) contient le widget *.params.l1*, la seconde (ligne 0, colonne 1) contiendra *.params.e1* et ainsi de suite. Les widgets sont collés à gauche (option *-sticky w*, w=west).

La commande **pack** permet l'affichage de la partie *.params*, également aligné à l'ouest (option *-anchor w*).

Dans le gestionnaire *Grid*, les cellules sont automatiquement adaptées à la taille des widgets. Un widget peut éventuellement couvrir plusieurs cellules.

Checkbutton widget

```
frame .options checkbutton .options.r -text {considers "end of records" c
     -variable opt_r -onvalue "-r" -offvalue ""
checkbutton .options.l -text {matches each line of the text independently
     -variable opt_l -onvalue "-l" -offvalue ""
checkbutton .options.c -text {counts the number successfully matched recc
     -variable opt_c -onvalue "-c" -offvalue ""
pack .options.r .options.l .options.c  -anchor w
```

La commande **checkbutton** crée des cases à cocher, une variable et une valeur sont associées au bouton.

Dans notre exemple on associe directement la valeur de l'option au bouton, ce qui facilitera le traitement ensuite.

Les cases à cocher sont incluses dans un *frame* nommé .*options*.

Boutons de pilotage

```
frame .button -relief sunken -borderwidth 4
button .button.r  -text "Run !" -command \
   {set comm "$grappe $opt_r $opt_l $opt_c \"$pattern\" [glob $file]"
    .res insert end "=============\n$comm \n=============\n"
    pack .button.d -after .button.q -side right
    res insert end "[eval exec $comm] \n" }
button .button.q -text "Fini !" -command {exit}
button .button.d -text "Efface !" \
  -command {.res delete 1.0 end ; pack forget .button.d}
pack .button.r .button.q -side left
```

Rien de bien nouveau ici : **exit** pour quitter le programme, **glob** pour trouver les noms des fichiers.

Décoration pour la **frame** .*button* : -*relief sunken* et -*borderwidth 4*.

Noter la récupération des options pour créer la commande, et le placement du résultat de la commande dans .*res* avec *insert end*.

Remarquer l'intelligence du bouton *Efface !*, disparaît (*pack forget*) après avoir effacé le résultat. **Belle ergonomie !!**

Widget Text

```
text .res
pack .options .button .res -fill x
proc bgerror {err} {
     bell
    .res insert end "Erreur : $e \nPas de résultat !\n"
}
```

text crée un *widget*, qui contient une zone de texte multiligne éditable de longueur quelconque. Cette zone se comporte comme un éditeur de texte (copier/coller, multiples styles, hypertexte, images, défilement).

Dans notre exemple, on se contente de créer une zone simple.

On affiche les dernières zones avec l'extension horizontale automatique (option -*fill x*).

Toujours **bgerror**, qui sonne les cloches place un message dans la zone résultat.

Début de TkGrappe

```
#!/bin/sh
# the next line restarts using wish \
exec wish "$0" "$@"

set grappe "../bin/grappe"
wm title . "Grappe Tcl/Tk Interface"
```

Les trois premières lignes, permettent de transformer un script **Tcl/Tk** en commande **Unix**.

grappe contient le chemin du programme **grappe**, si ce chemin change on change juste cette ligne dans le programme. C'est toujours bon de placer les constantes en début de programme.

La commande **wm** fait des appels au *window manager*, ici elle demande de modifier le titre de la fenêtre **.**, vous savez la racine de la hiérarchie.

Ces lignes sont à ajouter au début du programme.

Visualisation de graphiques

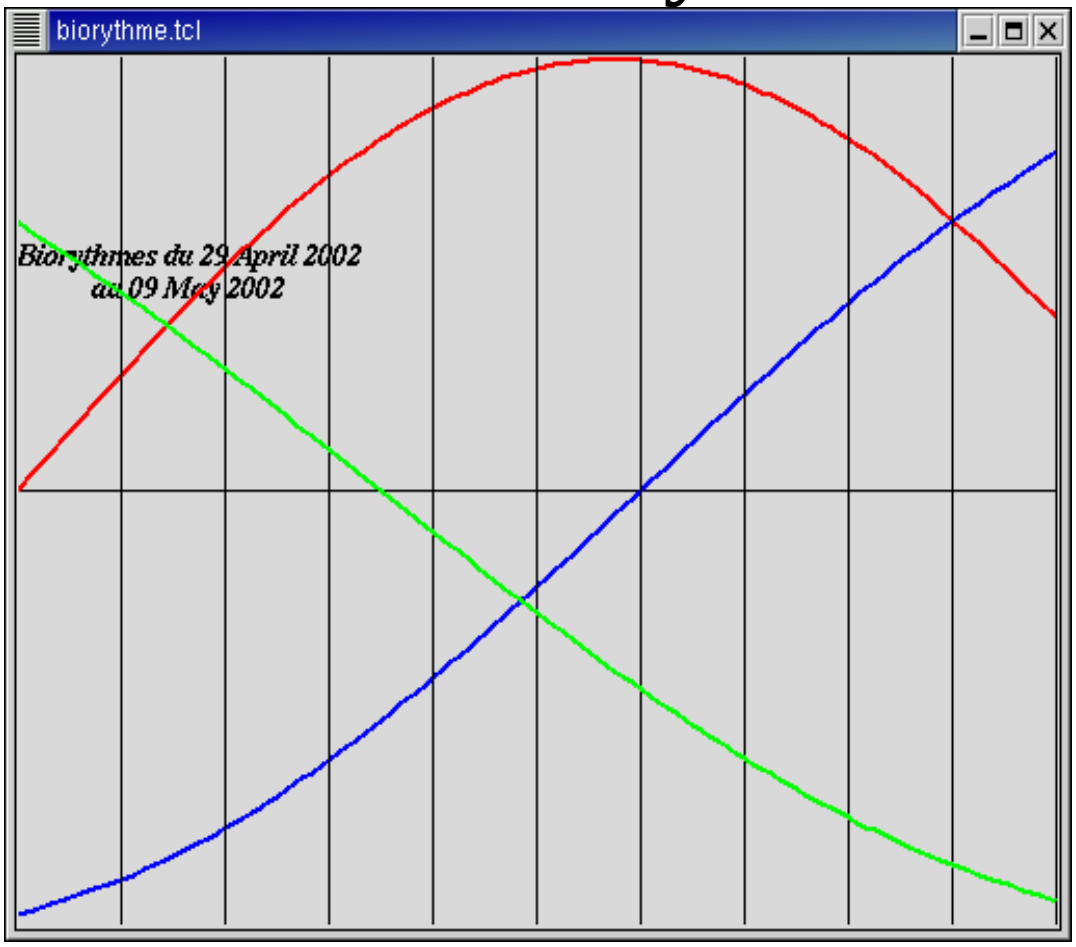
Dans cette section, on va découvrir un nouveau widget qui permet d'afficher des graphiques. Ce widget sera utilisé pour visualiser des sinusoïdes correspondant à des biorythmes.

Biorythme

Les biorythmes sont des cycles de vie, qui commencent à la naissance et varient péridodiquement. On distingue trois cycles : physique, émotif et mental de période 23, 28 et 33 jours.

Etant donné un jour de naissance, une date D et une durée d le programme a pour but de visualiser les biorythmes pendant la durée d depuis la date D.

Vue du biorythme



Code du dessin

```
canvas .c -height 400 -width 500
pack .c -expand 1

.c create line 0 200 500 200
.c create text 0 100 -text "Biorythmes"
    -anchor w -font "Times 12 bold italic"

foreach cy [array names bio] {
    set lcoords ""
    foreach point $bio($cy) {
        set x [expr [lindex $point 0]*500/([llength $bio($cy)]-1)]
        set y [expr 200-[lindex $point 1]*200]
        lappend lcoords $x $y
        .c create line $x 0 $x 400 }
    set id [.c create line $lcoords -width 2 -smooth 1 -fill $couleur($cy
    .c create text $x $y -text $cy -anchor e -fill $couleur($cy) \
 -font "Times 20 bold" -tag $cy -state hidden
    .c bind $id <Enter> ".c itemconfigure $cy -state normal"
    .c bind $id <Leave> ".c itemconfigure $cy -state hidden"
}
```

Widget Canvas

La commande *canvas* crée une zone de dessin dans la fenêtre d'affichage.

Dans le dessin on peut créer des objets graphiques, dans l'exemple on voit les objets suivants :

- Text : **create text** *coords*
- Ligne droite : **create line** *coords*
- Ligne courbe : **create line** *coords* **-smooth 1**

Aux objets graphiques peuvent être associés des callbacks sur évènement.
Des étiquettes (*tag*) sont associées aux objets permettant de manipuler un ensemble d'objets d'un coup.

Calcul du biorythme

```
proc biorythme {naissance date {duree 28}} {
   global cycle bio
   set pi2 [expr acos(-1)*2] *
   set jnai [clock scan $naissance]
   set jdat [clock scan $date]
   set d0 [expr ($jdat-$jnai)/(3600*24)]
   foreach cy [array names cycle] {
       set p $cycle($cy)
       set bio($cy) ""
       for {set j 0} {$j <= $duree} {incr j} {
         lappend bio($cy) "$j [expr sin(($pi2*(($j+$d0) % $p))/$p)]"
}  }   }
```

Pour chaque cycle cette procédure calcule, une liste des valeurs du biorythme et la stocke dans *bio($cy)*.

Perl/TK Introduction

http://www.perltk.org/

Perl est un langage de scripts, très efficace pour la recherche de formes dans des textes et les appels systèmes sous Unix. Il a été longtemps le langage de prédilection des développeurs de sites Web.

Perl/Tk est un module pour le langage Perl, qui implémente et permet l'utilisation des widgets Tk.

Programme de base Perl/Tk

```
use Tk;

my $mw = MainWindow->new;

my $titre = $mw->Label(-text=>"Conversion francs-euros\n saisissez la val
my $francw = $mw->Entry(-textvariable=>\$franc)->pack;
my $conv = $mw->Button(-text=>" conversion ",
          -command=>sub {$euro = $franc/6.55957})->pack;
my $eurow = $mw->Label(-textvariable=>\$euro)->pack;

MainLoop;
```

Chargement du module **Tk** se fait par l'appel de la primitive **use**

Création fenêtre racine par commande **MainWindow->new**.

Programme terminé par boucle d'évènements : **MainLoop;**. Classique dans les environnements graphiques. Automatique en Tcl.

Création de widgets en Perl

Widgets créés par envoi de messages sur objets graphiques.

Exemple : création d'un *label* par envoi du message *Label* sur fenêtre principal *$mw*.

```
my $titre = $mw->Label(-text=>"Conversion francs-euros");
```

Positionnement des objets également par envoi de message sur les objets, par exemple le message *pack*.

```
$titre->pack;
```

Paramètres des objets sous la forme *-attribut=>valeur*

Tk Variables

Perl/Tk autorise l'utilisation de variables **Perl** dans **Tk**. Lors de la déclaration d'une variable, on doit la préfixer par '\' afin qu'elle ne soit pas évaluée par **Perl**.

Exemple : association de la variable *$franc* au widget *$francw* puis utilisation dans une formule de calcul.

```
$francw = $mw->Entry(-textvariable=>\$franc);
$euro = $franc/6.55957;
```

Callback sur commande ou évènement

Partout où **Tk** demande un *callback*, il faut passer une fonction **Perl**.
Il est possible soit de passer directement le code de la fonction avec le mot-clé *sub* et un bloc de code, soit le nom de la fonction préfixé par '\' pour reporter l'évaluation.

Exemple 1 : le code de la conversion est directement associé à la commande

```
$conv = $mw->Button(-text=>" conversion ",
                    -command=>sub {$euro = $franc/6.55957});
```

Exemple 2 : le code de la conversion est défini dans la fonction *toeuro*.

```
sub toeuro {
    $euro = $franc/6.55957;
};
$mw->Button(-text=>" conversion ",
            -command=>\&toeuro};
```

Python/Tkinter Introduction

http://www.python.org/topics/tkinter/

Python est un langage de scripts mdoulaire et orienté objet. En fait Python est un langage générique du même niveau que Java, utilisant le typage dynamique.

Tkinter est un module pour le langage Python, qui implémente et permet l'utilisation des widgets Tk dans des programmes Python.

Programme de base Python

```
from Tkinter import *
tk = Tk()

(franc,euro) = (DoubleVar(),DoubleVar())
def seteuro():
    euro.set(franc.get()/6.55957)

Label(tk, text="Conversion francs-euros\n saisissez la valeur en francs."
Entry(tk, textvariable=franc).pack()
Button(tk, text="conversion", command=seteuro).pack()
Label(tk, textvariable=euro).pack()

tk.mainloop()
```

Chargement du module **Tkinter** par appel de **import** : *from Tkinter import **

Création fenêtre racine par commande **Tk()**.

Programme terminé par boucle d'évènements sur la fenêtre principale : **tk.mainloop()**. Classique on vous dit !

Création de widgets

Widgets obtenus par création d'objet de la classe du widget. Premier paramètre = widget parent.

Exemple : création d'un *label* par création d'un objet *Label()* dans fenêtre principal *tk*.

```
titre = Label(tk, text="Conversion francs-euros.")
```

Positionnement des objets par envoi de message sur les objets, par exemple le message *pack()*.

```
titre.pack()
```

Paramètres des objets sous la forme *attribut=valeur* (voir ci-dessus).

Tk Variables

Où **Tk** attend des variables, **Python** ne peut pas placer de variables, car cette notion n'existe pas. **Python** doit placer des objets de type *StringVar()*, *IntVar()* ou *DoubleVar()*.

Exemple : création et usage d'une variable nommée *franc* contenant un nombre flottant.

```
franc = DoubleVar()
Entry(tk, textvariable=franc)
```

Contenu d'une variable manipulé par les méthodes *get()* (lecture) et *set(valeur)* (modification).

Exemple : lecture de *franc*, et modification d'*euro* après calcul.

```
euro.set(franc.get()/6.55957)
```

Callback sur commande ou sur évènement

Partout où **Tk** demande un *callback*, il faut passer un objet *function* ou *method* de **Python**.

Tkinter appelle la commande associée à un widget sans paramètre. Quant aux liaisons sur évènements, les callbacks sont appelés avec un objet de type *event* en premier paramètre.

Exemple : association d'une commande *seteuro* à un bouton.

```
def seteuro():
    euro.set(franc.get()/6.55957)
conv = Button(tk, text="conversion", command=seteuro)
```

Exemple : liaison de l'évènement sur l'objet *weuro* à la commande *setfranc*.

```
def setfranc(e):
    franc.set(round(euro.get()*6.55957, 2))
weuro.bind('<KeyRelease>', setfranc)
```