Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PYTHON

Volume Dix

Parties 54 à 59

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

En réponse aux requêtes des lecteurs, nous avons réuni le contenu de certains articles consacrés à la programmation en python.

Pour l'instant, il s'agit d'une réédition directe de la série **Programmer en Python**, **parties 54 à 59**, des numéros 85, 86, 87, 91, 95 et 100, par l'incomparable professeur en Python Greg Walters.

Veuillez considérer l'origine de la publication ; les versions actuelles du matériel et des logiciels peuvent différer de ceux que nous présentons, ainsi vérifiez bien votre matériel et la version de vos logiciels avant d'émuler les tutoriels de cette édition spéciale. Vous pouvez installer des versions de logiciels plus récentes ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Sommaire

Programmer en Python - Partie 54

Écrit par Greg D. Walters

Il y a plusieurs années, j'avais quelques problèmes d'hypertension. Mon médecin m'a suggéré de trouver une activité qui m'obligerait à me concentrer sur quelque chose d'assez utile, mais plutôt simple. Je m'en suis sorti en essayant de faire du point de croix compté. C'est créatif, ciblé, et maintient votre esprit occupé sur ce que vous faites, pas sur ce qui vous dérange. Me trouvant à nouveau dans cette situation, j'ai ressorti le cerceau et les aiguilles et j'ai recommencé.

Au cas où vous ne seriez pas familier avec le point de croix compté, je vais vous donner un bref aperçu de ce que c'est. Le point de croix est un type de couture qui consiste à faire des « x » minuscules en fil pour aboutir au final à une image. Le fil s'appelle « floss » ou « soie » et le tissu que vous utilisez s'appelle « aïda ». Selon Wikipedia, l'aïda est un tissu spécial formé de petits carrés et de petits trous à intervalles réguliers. Ceci facilite le placement des motifs en « x » qui forment l'image. Il existe deux types de point de croix. L'un a une image imprimée sur la toile aïda (une sorte de peinture à numéros), et l'autre utilise une toile aïda totalement vierge et c'est vous qui comptez les points du motif. La seconde est beaucoup plus compliquée que la première. Allez dans votre magasin de tissu préféré ou à la section loisirs créatifs de votre hypermarché local et vous comprendrez.

J'ai également commencé à m'amuser il y a quelque temps avec la création d'un programme qui prendrait une image pour la convertir en un modèle de point de croix. Une chose en entraînant une autre, j'ai dû laisser le programme de côté pour faire d'autres choses. J'ai maintenant déterré l'idée et commencé à nouveau.

Nous allons traiter ce projet durant les prochains articles. Il faudra néanmoins du temps, car certaines choses sont assez complexes et comprennent de nombreuses parties. Voici le « plan de jeu » :

- Créer une base de données pour convertir les couleurs de pixels en couleurs de fils.
- Créer une interface graphique avec Tkinter pour l'application.
- Étoffer l'application pour faire la manipulation des images.
- Créer un fichier PDF qui sera le modèle ultime pour le projet.

Ce que vous apprendrez :

- révision sur la manipulation de base de données et XML ;
- révision sur la programmation d'interfaces avec Tkinter. Si vous avez raté les précédents articles à ce sujet, merci de vous référer aux numéros 51 à 54 ;
- manipulation d'images avec PIL (http://pillow.readthedocs.org/en/latest/) ;
- création de PDF à l'aide de pyFPDF (https://code.google.com/p/pyfpdf).

Pour commencer

La première chose dans notre liste de tâches est de créer la base de données qui contiendra les couleurs de fils DMC (™) et de les rapprocher le mieux possible des valeurs RVB (Rouge, Vert, Bleu) utilisées pour les images sur ordinateur. La base de données contiendra également la valeur hexadécimale et la représentation HSV (Teinte, Saturation, Lumière) pour chaque couleur de fil. Il semble que le HSV soit la façon la plus simple de trouver la « plus proche » représentation d'une couleur de fil. Bien sûr, l'œil humain est le décideur ultime. Si vous n'êtes pas familier avec les représentations de couleurs HSV, il y a un article assez complexe sur Wikipedia : http://fr.wikipedia.org/wiki/Teinte_saturation_lumi%C3%A8re. Il pourrait aider, mais il pourrait rendre les choses moins claires.

La première chose dont nous avons besoin est un fichier XML qui contient les couleurs de fils DMC avec une conversion RVB. Le meilleur que j'ai trouvé est ici : http://sourceforge.net/p/kxstitch/feature-requests/9/. Le fichier que vous cherchez est dmc.xml. Téléchargez-le et mettez-le dans le dossier que vous utiliserez pour stocker le code Python.

Maintenant, nous allons utiliser apsw (ci-dessous), que vous devriez déjà avoir, pour manipuler la base de

```
# makedb.py
# DMC.xml vers SQLite database
# Pour Full Circle Magazine numero 85

import apsw
from xml.etree import ElementTree as ET
nomtable = "DMC"
```

données et ElementTree pour faire l'analyse XML (inclus depuis Python version 2.7).

Comme toujours, nous commençons par nos importations. Dans ce programme, nous n'en avons que deux. Nous réglons également le nom de la table.

La partie suivante devrait vous être familière si vous avez lu les articles depuis un certain temps. Nous créons une fonction pour lire le fichier XML et l'analyser. Nous pouvons ensuite utiliser ces informations pour remplir la base de données. Un extrait du fichier XML est affiché en haut à droite.

Nous recherchons la balise <floss> pour chaque ligne d'information. Pour ce faire, nous utilisons la commande .findall('floss'). Une fois que nous avons la ligne d'information, nous découpons chaque balise (nom, description, etc.) en variables distinctes pour les placer dans la base de données. Quand nous arrivons à la balise <color>, nous utilisons la commande .floss.findall('color') pour obtenir chaque valeur de Rouge, Vert et Bleu.

Nous commençons par indiquer à la fonction que nous allons utiliser les variables globales connexion et curseur. Nous indiquons ensuite le nom du fichier XML, analysons le fichier XML, et démarrons. Nous utilisons également une variable compteur pour montrer que quelque chose se passe durant les analyses et insertions en base de données.

Maintenant que nous avons toutes nos données, nous devons créer l'instruction d'insertion SQL et l'exécuter. Notez le « \ » après le mot VALUES dans l'instruction SQL. C'est un caractère de continuation de ligne pour faciliter l'impression ici dans le magazine. Nous allons créer la base de données et une table dans quelques instants.

```
  SQL = "INSERT INTO DMC
(DMC,Description,Rouge,Vert,B
leu) VALUES \

('%s','%s',%s,%s,%s)" %
(nom,desc,rouge,vert,bleu)

 cursor.execute(SQL)
```

Maintenant, nous affichons dans la fenêtre de terminal que quelque chose se passe :

```
   print "Enregistrement en
cours : {0}".format(compteur)

   compteur += 1
```

Maintenant, nous créons et/ou ouvrons la base de données dans la routine OuvrirBase (à droite en bas).

```
<floss>
    <name>150</name>
    <description>Dusty Rose Ultra VDK</description>
    <color>
         <red>171</red>
         <green>2</green>
        <blue>73</blue>
    </color>
</floss>
```

```
def LireXML():
  global connexion
  global curseur
  nomfic = 'dmc.xml'
  arbre = ET.parse(nomfic)
  racine = arbre.getroot()
  compteur = 0
  for fil in racine.findall('floss'):
      nom = fil.find('name').text
      desc = fil.find('description').text
      for couleur in fil.findall('color'):
          rouge = couleur.find('red').text
          vert = couleur.find('green').text
          bleu = couleur.find('blue').text
```

```
def OuvrirBase():
   global connexion
   global curseur
   global ucurseur
   global nombase
   connexion = apsw.Connection("fils.db3")
   curseur = connexion.cursor()
   ucurseur = connexion.cursor()
```

Si vous étiez avec nous quand nous avons travaillé avec les bases de données, vous avez remarqué que nous utilisons deux curseurs cette fois. La variable curseur est utilisé pour les insertions « normales », et plus tard dans l'instruction select pour la mise à jour afin de régler les valeurs hex et HSV. Nous devons utiliser deux curseurs, car si vous modifiez un curseur au milieu d'une instruction logique, vous perdez tout avec une nouvelle commande. Nous pouvons utiliser « ucursor » pour les déclarations de mise à

jour. À part cela, il s'agit de notre routine OpenDB habituelle.

Maintenant que la base de données est créée et/ou ouverte, nous pouvons mettre en place notre table (en haut à droite). Notez que l'instruction SQL en haut à droite utilise les guillemets triples pour permettre un saut de ligne pour la lisibilité.

La routine ViderTables (au milieu à droite) est là juste pour s'assurer que si nous voulons ou devons exécuter l'application plus d'une fois, nous commençons avec une table vide et propre si elle existe.

Si nous devions nous arrêter ici, nous aurions une base de données de travail raisonnable avec les couleurs DMC, leur nom et les valeurs RVB associées à chacune. Cependant, comme je l'ai mentionné précédemment, il est plus facile de choisir la couleur de fil la plus proche en utilisant les données HSV.

Nous créons donc ensuite la valeur hexadécimale pour les valeurs RVB (au milieu à gauche).

La fonction suivante crée les valeurs HSV à partir des valeurs RVB. J'ai trouvé l'algorithme sur internet. Vous pouvez le chercher aussi.

```
def CreerTables():
  sql = '''CREATE TABLE IF NOT EXISTS DMC
          (pkID INTEGER PRIMARY KEY, DMC INTEGER,
          Description TEXT, Rouge INTEGER, Vert INTEGER, Bleu INTEGER,
          HEX TEXT,H INTEGER,S INTEGER,V INTEGER)'''
  curseur.execute(sql)
```

```
def rgb2hex(rgb):
    return '%02x%02x%02x' % rgb
```

```
def ViderTables():
 sql="DELETE FROM %s" % nomtable
 curseur.execute(sql)
```

Enfin, nous créons la fonction MAJBase (à voir sur Pastebin). Nous utilisons la commande SELECT * FROM DMC et la variable de curseur « standard » pour contenir les données. Nous parcourons ensuite les données récupérées, lisons les valeurs RVB et les transmettons à la fonction rgb2hex comme un tuple et à la fonction rgb2hsv comme trois valeurs distinctes. Une fois que nous avons obtenu les valeurs de retour, nous utilisons la commande SQL update pour trouver l'enregistrement approprié en utilisant la clé primaire (pkID). Comme je l'ai déjà dit, nous devons utiliser un curseur distinct pour l'instruction de mise à jour.

La dernière chose à faire est d'appeler chacune des fonctions afin de créer la base de données et, à la fin, nous affichons « Fin » pour informer l'utilisateur que tout est fait.

```
OuvrirBase()
CreerTables()
ViderTables() # Juste pour
etre sur
LireXML()
```

```
def rgb2hsv(r, g, b):
    r, g, b = r/255.0, g/255.0, b/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) % 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) % 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) % 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
        v = mx
    return int(round(h,0)), int(round(s*100,0)),
int(round(v*100,0))
```

```
MAJBase()
print "Fin"
```

J'ai nommé ce programme « MakeDB ». La base de données devrait être créée dans le dossier où se trouve le code et le fichier XML. Comme toujours, le code complet peut être trouvé sur : http://pastebin.com/P7gZgNTm (http://pastebin.com/Zeqqw3pi pour le code en anglais).

La prochaine fois, nous travaillerons sur l'interface graphique en utilisant Tkinter, donc, en attendant, vous pouvez vous rafraîchir la mémoire en regardant les FCM nos 51 à 54 où je vous expliquais Tkinter.

Passez un bon mois. À la prochaine fois.

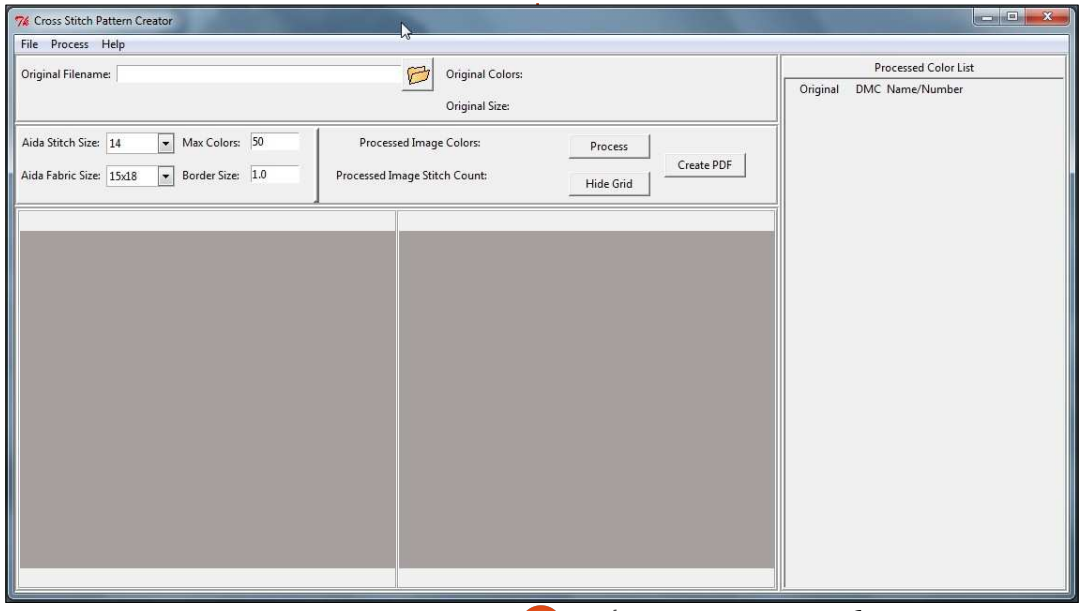
Programmer en Python - Partie 55

Écrit par Greg D. Walters

Voici la deuxième partie d'un long tutoriel sur la création d'un générateur de motifs de point de croix. Dans la première partie (le FCM n° 85), nous avons créé une base de données contenant les couleurs de fils DMC™ avec leurs valeurs RVB les plus proches. Dans cette partie, nous allons créer l'interface graphique en utilisant Tkinter. Nous allons également utiliser PIL (Python Imaging Library) et PMW (Python Mega Widgets). Vous aurez besoin de télécharger les bibliothèques et les installer avant d'aller plus loin. Pour PIL, récupérez la dernière version du fork Pillow sur https://github.com/python-imaging/Pillow. Pour PMW, téléchargez-le sur http://pmw.sourceforge.net/.

Vous aurez également besoin de deux fichiers images. L'un est un simple rectangle gris de 500×400 pixels. Vous pouvez utiliser GIMP ou un autre programme de manipulation d'images pour le créer. Nommez-le default.jpg, et placez-le dans votre répertoire de code source avec la base de données. L'autre est une image d'un dossier pour le bouton d'ouverture d'image. J'ai cherché le mot « folder » sur openclipart. J'en ai trouvé un pas mal ici : https://openclipart.org/detail/177890/file-folder-by-thebyteman-177890. Ouvrez-le dans GIMP, redimensionnez-le à 30×30 et enregistrez-le dans le même répertoire que les deux autres fichiers en tant que « open.gif».

Ci-dessous une capture d'écran de ce à quoi ressemblera l'interface graphique terminée. Il y a quatre fenêtres principales : trois sur le côté gauche et une à droite. Lorsque nous suivrons le processus de construction des widgets, je les nommerai fenêtre du haut, fenêtre du milieu, fenêtre du bas et fenêtre de côté. La fenêtre du haut contient l'image originale. La fenêtre du milieu sert au traitement de l'image. La fenêtre du bas montre l'image originale sur la gauche et l'image traitée sur la droite, et la fenêtre de côté affiche les couleurs et fils nécessaires. Il semble au premier abord qu'il y ait beaucoup d'espace perdu, mais quand vous verrez le programme fonctionner, il n'y aura pas tant d'espace vide que ça, une fois qu'on arrive à la partie de traitement.



Maintenant, nous pouvons commencer à travailler sur le code. Voici notre longue liste des importations…

```
from Tkinter import *
import tkFileDialog
import tkCommonDialog
import tkMessageBox
import ttk
from PIL import
Image,ImageTk,ImageOps
import Pmw
import apsw # Database
Access
import math # Math library
import sys
```

En voyant toutes ces importations, vous vous doutez que cela va être un long programme. En fait, la partie de code pour l'interface utilisateur fera plus de 300 lignes, commentaires compris. La « bonne » nouvelle c'est qu'environ 200 de ces lignes concernent la partie Tkinter du programme, l'interface graphique elle-même. La plupart des lignes restantes dans cette partie sont les préparatifs pour les fonctions du prochain article.

Nous allons créer une classe pour contenir tout le code de l'interface utilisateur (page suivante, en haut à droite).

Nous avons d'abord la définition de la classe et à côté nous avons la fonction _init_ à laquelle nous passons la fenêtre « racine ». Nous créons la fenêtre racine dans les quatre dernières lignes du programme. Dans la fonction _init_ nous définissons toutes les variables globales et faisons quelques affectations initiales avant de commencer les autres fonctions. La première chose que nous faisons est de créer une liste de tuples qui contiennent les formats de fichiers images dont nous avons besoin lorsque nous appelons le dialogue OpenFile. Les deux lignes suivantes ci-dessous définissent et préparent les deux fichiers images que nous venons de créer (le fichier GIF de dossier ouvert et le rectangle gris, qui seront utilisés comme des espaces réservés pour nos images utilisées pour créer le motif).

```
self.openimage =
PhotoImage(file='open.gif')

self.DefaultImage
=ImageTk.PhotoImage(self.Thum
bnail("default.jpg",450,450))
```

Maintenant nous entrons dans les définitions globales (au milieu à droite). Vous vous souvenez peut-être que lorsque vous utilisez Tkinter, si vous avez un widget comme une boîte de saisie de texte ou une liste déroulante et que vous souhaitez récupérer les informations sélectionnées ou saisies, vous définissez une variable globale, puis l'assignez à une classe de variables (BooleanVar, DoubleVar, IntVar ou StringVar). Elle « pistera » alors les modifications dans les valeurs du widget afin que vous puissiez y accéder avec les méthodes .get() ou .set(). Dans les prochaines lignes de code, nous créons le nom de la variable globale, puis l'affectons à la classe correspondante. J'ai mis quelques commentaires dans le code pour essayer de vous aider à suivre ce que nous faisons.

Comme vous pouvez le voir, nous créons des variables : NomFichierOriginal, qui contient l'image à partir de laquelle nous voulons créer le motif, NombreCouleursOriginal qui détient le nombre de couleurs de l'original et TailleOriginal qui détient la taille en pixels de l'original. Comme ils disent à la télé… « ***Mais attendez, il y en a encore plus !*** » (à droite).

La variable ComboTaillePoints est réglée par une liste déroulante et gère la taille des points de la toile aïda que vous souhaitez utiliser pour votre projet. La variable ComboTaille est également définie par une zone de liste déroulante et contient la taille de la toile aïda. LargeurTissu et HauteurTissu sont les dimensions de la toile aïda. MaxCouleurs est réglée à partir d'une zone de saisie pour définir le nombre de couleurs et TailleBordure est une valeur en virgule flottante qui indique la quantité d'aïda utilisé pour le cadre.

```
class XStitch:
    def __init__(self, principal):
        self.formatsImages = [
        ('JPEG / JFIF','*.jpg'),
        ('Portable Network Graphics','*.png'),
        ('CompuServer GIF','*.gif'),
        ('Windows Bitmap','*.bmp'),
        ('Tous les types *.*','*.*'),
        ]
```

```
        #----------------------------------------
        #               Definitions globales
        #----------------------------------------
        #           pour l'interface graphique
        global NomFichierOriginal
        NomFichierOriginal = StringVar()
        global NombreCouleursOriginal
        NombreCouleursOriginal = StringVar()
        global TailleOriginal
        TailleOriginal=StringVar()
```

```
        global ComboTaillePoints
        ComboTaillePoints = IntVar()
        global ComboTaille
        ComboTaille = StringVar()
        global LargeurTissu
        LargeurTissu = DoubleVar()
        global HauteurTissu
        HauteurTissu = DoubleVar()
        global MaxCouleurs
        MaxCouleurs = IntVar()
        global TailleBordure
        TailleBordure = DoubleVar()
```

```
global CouleursTraitees

CouleursTraitees =
StringVar()

global TailleTraitee

TailleTraitee = StringVar()

global CouleurDMC

CouleurDMC = StringVar()
```

Les dernières variables de type « classes de variables » sont utilisées pour les informations une fois que nous avons traité l'image originale avec les paramètres souhaités.

La prochaine série de variables globales (en haut à droite) est utilisée pour faciliter l'accès tout au long du programme. Pour la plupart, leur nom est explicite, ou le deviendra une fois que nous les utiliserons. Il y a trois variables pas si évidentes. couleurFond1 et couleurFond2 sont des tuples utilisés dans le processus de maillage, et la variable PretPourTraitement est utilisée pour indiquer que l'image d'origine est chargée et que tout est prêt pour commencer – juste au cas où l'utilisateur appuie sur le bouton Traitement trop tôt.

Voilà, nous avons créé toutes nos variables globales et arrivons au code qui crée réellement l'interface graphique. Nous ouvrons la base de données, créons le menu, mettons en place les widgets et enfin plaçons les widgets aux endroits appropriés. Juste pour vous donner un aperçu, nous utiliserons le gestionnaire de placement en grille. Nous verrons cela plus tard.

```
#------------------------

self.OuvrirBase()
self.FabriquerMenu(principal)
frm =
self.ConstruireWidgets(principal)

self.PlacerWidgets(frm)
```

La prochaine partie de notre code (au milieu à droite) met en place la barre de menu. J'ai essayé de rester logique pour qu'il soit facile à comprendre.

Nous définissons une fonction appelée FabriquerMenu, avec pour argument la fenêtre racine. Nous définissons ensuite les trois jeux de menus que nous allons créer. Un menu Fichier, un pour le traitement et le dernier pour l'aide.

```
menu.add_cascade(label="Fichie
r", menu=menuFichier)

menu.add_cascade(label="Tr
aitement",menu=Traitement)

menu.add_cascade(label="A
ide",menu=Aide)
```

Maintenant, nous mettons en place les options du menu Fichier (à droite). Ouvrir permet d'ouvrir notre image et utilise une fonction appelée RecupererNomFichier. Sauver va créer le fichier PDF de sortie et utilise la fonction SauverFichier. Nous ajoutons un séparateur et enfin une ligne pour Quitter.

Maintenant, nous avons l'option de traitement et les fonctions d'aide (page suivante, en haut à droite).

Toutes les options de la barre de menu sont également disponibles à partir de divers boutons dans le programme. Maintenant, nous allons écrire notre fonction ConstruireWidgets. C'est là que nous créons tous les widgets qui seront utilisés sur l'interface graphique.

```
        #----------------------------------------
        global AfficherGrille
        AfficherGrille = True
        global ImageTraitee
        ImageTraitee = ""
        global GrilleImage
        GrilleImage = ""
        global couleurFond1
        couleurFond1 = (120,)*3
        global couleurFond2
        couleurFond2 = (0,)*3
        global PretPourTraitement
        PretPourTraitement = False
```

```
#=====================================================
#             DEBUT DEFINITION INTERFACE
#=====================================================
    def FabriquerMenu(self,principal):
            menu = Menu(principal)
            racine.config(menu=menu)
            menuFichier = Menu(menu, tearoff=0)
            Traitement = Menu(menu,tearoff=0)
            Aide = Menu(menu,tearoff=0)
```

```
#----------------------------------------
#               Menu Fichier
#----------------------------------------
menuFichier.add_command(label="Nouveau")
menuFichier.add_command(label="Ouvrir", command=self.RecupererNomFichier)
menuFichier.add_command(label="Sauver", command=self.SauverFichier)
menuFichier.add_separator()
menuFichier.add_command(label="Quitter", command=self.Quitter)
```

```
def
ConstruireWidgets(self,principal):

      self.frame =
Frame(principal,width=900,hei
ght=850)
```

Nous commençons par la définition de la fonction (en bas à droite), qui prend en argument la fenêtre racine (principale) et crée un cadre qui contient tous nos autres widgets. J'ai ajouté des commentaires pour aider à comprendre quelle partie du code traite de quelle fenêtre. Nous allons commencer avec la fenêtre supérieure.

En supposant que vous vous en souvenez ou avez rafraîchi votre mémoire sur Tkinter, cela devrait être assez simple. Regardons la première étiquette pour l'expliquer :

```
self.label1 =
Label(self.frm1,text =
“Fichier original : ”)
```

Premièrement, nous définissons le nom du widget (self.label1 =). Ensuite, nous réglons cette variable au type de widget que nous voulons utiliser ; dans ce cas Label (étiquette). Enfin, nous définissons les paramètres que nous voulons appliquer à ce widget, à commencer par le widget parent (self.frm1) et, dans ce cas, le texte qui apparaîtra sur l'étiquette. Maintenant, nous allons prendre un moment pour regarder le bouton self.btnNomFic.

```
self.btnNomFic =
Button(self.frm1, width=28,
image=self.openimage,

command=self.RecupererNomFichier)
```

La première chose à remarquer est que c'est scindé en deux lignes. Vous pouvez placer le tout sur une seule ligne sans problème… mais c'est tout simplement trop long pour tenir sur une ligne de 72 caractères. Nous allons vraiment faire attention aux paramètres que nous utilisons ici. D'abord, le parent (frm1), puis la largeur qui est fixée à 28. Lorsque nous utilisons un widget qui a l'option de contenir un texte ou une image, il faut faire attention au réglage de la largeur. S'il contient du texte, le paramètre de largeur représente le nombre de caractères qu'il contiendra. Si c'est pour afficher une image, il correspond au nombre de pixels. Enfin, nous réglons le paramètre de commande, qui indique au système quelle fonction appeler lorsque le bouton est cliqué.

```
#----------------------------------------
#           Menu Traitement
#----------------------------------------
Traitement.add_command(label="Tous",command=self.Traitement)
#----------------------------------------
#               Menu Aide
#----------------------------------------
Aide.add_command(label="Aide",command=self.AfficherAide)
Aide.add_separator()
Aide.add_command(label="A propos",command=self.AfficherApropos)
```

Une autre chose à regarder est le paramètre textvariable. Il indique la variable qui contiendra l'information qui sera affichée dans le widget. Nous avons réglé ces variables dans la fonction _init_ plus tôt. Une autre chose à mentionner est que le cadre lui-même a deux paramètres que vous pourriez oublier. Le paramètre Relief définit le type de bordure, qui dans ce cas est GROOVE, et le paramètre bd définit la largeur de la bordure. La largeur de la bordure vaut 0 par défaut, donc si vous voulez voir l'effet, vous devez

```
# --------------- FENETRE HAUT --------------------
self.frm1 = Frame(self.frame,width=900,height=100,bd=4,relief=GROOVE)
self.label1 = Label(self.frm1,text = "Fichier original : ")
self.entNomFic = Entry(self.frm1,width=50,textvariable=NomFichierOriginal)
self.btnNomFic = Button(self.frm1,width=28,image=self.openimage,command=self.RecupererNomFichier)
self.label2 = Label(self.frm1,text = "Nb couleurs de l'original : ")
self.lblNombreCouleursOriginal = Label(self.frm1,text="",width=10,textvariable=NombreCouleursOriginal)
self.label3 = Label(self.frm1,text = "Taille de l'original : ")
self.lblTailleOriginal = Label(self.frm1,text="",width=10,textvariable=TailleOriginal)
```

```
# --------------- FENETRE MILIEU --------------------
self.frm2 = Frame(self.frame,width=900,height=160,bd=4,relief=GROOVE)
self.lbl4 = Label(self.frm2,text="Taille des points de l'aida : ")
self.lbl5 = Label(self.frm2,text="Taille du tissu Aida : ")
self.TCombobox1 = ttk.Combobox(self.frm2,textvariable=ComboTaillePoints,width=8)
self.TCombobox1.bind('<<ComboboxSelected>>', self.ChoixTaillePoints)
self.TCombobox1['values'] = (7,10,11,12,14,16,18,22)
self.TCombobox2 = ttk.Combobox(self.frm2,textvariable=ComboTaille,width = 8)
self.TCombobox2.bind('<<ComboboxSelected>>',self.ChoixTailleAida)
self.TCombobox2['values'] = ("12x18","15x18","30")
```

définir la largeur de bordure (bd est un raccourci).

Maintenant, nous allons nous occuper des widgets de la fenêtre du milieu.

Les six dernières lignes de cette section (page précédente, en bas à droite) gère les deux listes déroulantes de l'interface utilisateur. Chaque liste déroulante est sur trois lignes (je les ai écrites ainsi pour les rendre faciles à comprendre). La première ligne contient les paramètres de base. Sur la ligne suivante, nous relions l'événement de « changement de choix » à la fonction ChoixTaillePoints, et la dernière ligne contient la liste des valeurs disponibles dans le menu déroulant.

Tout le reste ci-dessus est assez « classique ». Maintenant, nous réglons nos valeurs par défaut pour les widgets qui en ont besoin. Encore une fois, nous utilisons les variables globales mises en place dans la fonction _init_ et associées aux classes de variables de widgets.

```
ComboTaillePoints.set(14)
ComboTaille.set("15x18")
LargeurTissu.set(15)
HauteurTissu.set(18)
MaxCouleurs.set(50)
TailleBordure.set(1.0)
```

```
self.lbl6 = Label(self.frm2,text="Nb max de couleurs : ")
self.entMaxCouleurs = Entry(self.frm2,textvariable=MaxCouleurs,width=3)
self.lbl7 = Label(self.frm2,text="Taille bordure : ")
self.entTailleBordure = Entry(self.frm2,textvariable=TailleBordure,width = 8)
self.frmLine = Frame(self.frm2,width=6,height=80,bd=3,relief="raised")
self.lbl8 = Label(self.frm2,text="Couleurs traitees : ")
self.lbl9 = Label(self.frm2,text="Nb points traites : ")
self.lblCouleursTraitees=Label(self.frm2,width=10,textvariable=CouleursTraitees,justify=LEFT)
self.lblTailleTraitee=Label(self.frm2,width=10,textvariable=TailleTraitee,justify=LEFT)
self.btnDoIt = Button(self.frm2,text="Traitement",width=11,command = self.Traitement)
self.btnAfficherGrille = Button(self.frm2,text="Masquer grille",width=11,command=self.AfficherMasquerGrille)
self.btnCreerPDF=Button(self.frm2,text="Creer PDF",width=11,command=self.CreerPDF)
```

```
        # --------------- FENETRE BAS ----------------------
        self.frm3 = Frame(self.frame,width=450,height=450,bd=4,relief=GROOVE)
        self.lblImageL =
Label(self.frm3,image=self.DefaultImage,height=400,width=400,borderwidth=2,relief=GROOVE)
        self.lblImageR =
Label(self.frm3,image=self.DefaultImage,height=400,width=400,borderwidth=2,relief=GROOVE)
```

Maintenant, nous gérons la fenêtre du bas. C'est très simple, puisque nous n'avons à mettre en place que le cadre et deux étiquettes que nous allons utiliser pour contenir nos images.

Enfin, nous traitons la fenêtre latérale, qui contiendra une ScrolledFrame (fenêtre à ascenseurs) de la bibliothèque PMW. C'est vraiment facile à utiliser et fournit une interface agréable pour l'information sur les fils qui devront être utilisés. Vous pouvez vous documenter vous-mêmes sur la ScrolledFrame, car nous avons encore beaucoup à faire ici.

C'est tout pour les widgets. Maintenant, nous devons les placer. Comme je l'ai dit plus tôt, nous utiliserons le gestionnaire en « grille », plutôt que les gestionnaires « absolu » ou « paquet ».

La méthode Grille place les wid-

```
        #---------------- FENETRE COTE ----------------------
        self.frm4 = Frame(self.frame,width = 300,height=580,bd=4,relief=GROOVE)
        # Cree la fenetre deroulante
        self.sf = Pmw.ScrolledFrame(self.frm4,
             labelpos = 'n', label_text = 'Liste couleurs traitees',
             usehullsize = 1,
             hull_width = 300,
             hull_height = 567,)
        return self.frame
```

gets (vous l'aurez deviné) sur une grille, par rangées et colonnes. Je vais utiliser la fenêtre supérieure à titre d'exemple (illustré en haut à droite).

D'abord, nous plaçons le cadre.

Vous pouvez voir que nous plaçons le widget en utilisant la commande {nomwidget}.grid, puis la position de ligne et de colonne. Notez que nous indiquons au widget de saisie de couvrir 5 colonnes. Les valeurs padx et pady mettront un espace supplémentaire à la fois sur les côtés droit et gauche (padx) et haut et bas (pady). Le paramètre sticky est similaire à une commande « justifier » pour le texte.

La fenêtre du milieu est un peu plus compliquée, mais similaire à celle du haut. Vous remarquerez peut-être un cadre supplémentaire au milieu du code (self.frmLine). Cela nous donne un beau séparateur entre la section des options et la section d'affichage. Comme il n'y a pas de widget ligne horizontale ou verticale, j'ai triché et utilisé un cadre d'une largeur de 6 pixels et une largeur de bordure de 3, ce qui le fait ressembler à une ligne épaisse.

La fenêtre du bas est simple puisque nous n'avons que le cadre et les deux étiquettes pour contenir les images.

```
LIGNE | Col 0  | Col 1 - Col 6          |Col 7    | Col 9 | Col 10                  |
      ------------------------------------------------------------------------------
 0    | Label1 | entNomFic              |btnNomFic| Label2|lblNombreCouleursOriginal|
 1    |                                           | Label3|lblTailleOriginal        |
      ------------------------------------------------------------------------------
```

```
    def PlacerWidgets(self,fenetre):
        fenetre.grid(column = 0, row = 0)
        # --------------- FENETRE HAUT ---------------------
        self.frm1.grid(column=0,row=0,rowspan=2,sticky="new")
        self.label1.grid(column=0,row=0,sticky='w')
        self.entNomFic.grid(column=1,row=0,sticky='w',columnspan = 5)
        self.btnNomFic.grid(column=7,row = 0,sticky='w')
        self.label2.grid(column=9,row=0,sticky='w',padx=10)
        self.lblNombreCouleursOriginal.grid(column=10,row=0,sticky='w')
        self.label3.grid(column=9,row=1,sticky='w',padx=10,pady=5)
        self.lblTailleOriginal.grid(column=10,row=1,sticky='w')
```

```
        # --------------- FENETRE MILIEU -------------------
        self.frm2.grid(column=0,row=2,rowspan=2,sticky="new")
        self.lbl4.grid(column=0,row=0,sticky="new",pady=5)
        self.lbl5.grid(column=0,row=1,sticky="new")
        self.TCombobox1.grid(column=1,row=0,sticky="new",pady=5)
        self.TCombobox2.grid(column=1,row=1,sticky="new")
        self.lbl6.grid(column=2,row = 0,sticky="new",padx=5,pady=5)
        self.entMaxCouleurs.grid(column=3,row=0,sticky="new",pady=5)
        self.lbl7.grid(column=2,row=1,sticky='new',padx=5)
        self.entTailleBordure.grid(column=3,row=1,sticky='new')
        self.frmLine.grid(column=4,row=0,rowspan=2,sticky='new',padx=15)
        self.lbl8.grid(column=5,row=0,sticky='new',pady=5)
        self.lbl9.grid(column=5,row=1,sticky='new')
        self.lblCouleursTraitees.grid(column=6,row=0,sticky='w')
        self.lblTailleTraitee.grid(column=6,row=1,sticky='new')
        self.btnDoIt.grid(column=7,row=0,sticky='e',padx=5,pady = 5)
        self.btnAfficherGrille.grid(column=7,row=1,sticky='e',padx=5,pady = 5)
        self.btnCreerPDF.grid(column=8,row=0,rowspan=2,sticky='ew',padx=10)
```

La fenêtre de côté est à peu près identique, sauf que la ScrolledFrame permet qu'un cadre soit défini à l'intérieur du widget. Nous créons ensuite

```
        # --------------- FENETRE BAS ----------------------
        self.frm3.grid(column=0,row=4,sticky="nsew")
        self.lblImageL.grid(column=0,row=0,sticky="w")
        self.lblImageR.grid(column=1,row=0,sticky="e")
```

trois widgets et les plaçons dans la grille comme des en-têtes de colonnes. Nous faisons cela, car nous avons réglé le cadre intérieur de la ScrolledFrame et nous devons assigner le parent (self.sfFrame) après sa création.

Le travail difficile est fini pour le moment. Maintenant nous allons créer toutes les fonctions dont nous aurons besoin pour obtenir une interface graphique qui fonctionne ; la plupart seront vides jusqu'au mois prochain. Nous en écrirons certaines, mais elles sont assez courtes.

La première fonction sera l'option Quitter de la barre de menu, dans le menu Fichier.

```
def Quitter(self):
    sys.exit()
```

La seule autre est la fonction Apercu. Nous en avons besoin pour remplir les rectangles gris dans les étiquettes de la fenêtre du bas. Nous lui passons le nom du fichier et la largeur et la hauteur que nous souhaitons pour l'aperçu.

Comme cet article est déjà long, je vais vous donner une liste de noms de fonctions et tout ce que vous avez à faire est de les remplir en utilisant la commande pass. Nous les compléterons le mois prochain. Je vais vous donner le premier comme un exemple, mais vous devriez déjà savoir comment le faire.

```
    def
RecupererNomFichier(self):
      pass
```

Pour le reste des fonctions, je vais vous donner les lignes def. Assurez-vous de tous les inclure dans votre code.

Comme vous pouvez le voir, nous avons pas mal de travail à faire le mois prochain. Nous avons encore quatre lignes à écrire pour finir pour ce mois, en dehors de notre code de classe :

```
racine = Tk()

racine.title("Generateur de
motifs de point de croix")

test = XStitch(racine)

racine.mainloop()
```

La première ligne met en place la fenêtre racine. La ligne suivante définit le titre sur la ligne supérieure. La troisième ligne instancie notre classe XStitch, et la dernière ligne démarre la boucle principale qui montre l'interface utilisateur et lui donne le contrôle.

Eh bien, ça fait beaucoup pour ce mois-ci, mais nous sommes arrivés au bout. Vous pouvez effectivement exécuter le programme pour voir l'interface graphique.

```
# ———— FENETRE COTE ————
self.frm4.grid(column=2,row=0,rowspan=12,sticky="new")
self.sf.grid(column=0,row=1)
self.sfFrame = self.sf.interior()
self.lblch1 = Label(self.sfFrame,text=" Original")
self.lblch2 = Label(self.sfFrame,text=" DMC")
self.lblch3 = Label(self.sfFrame,text="Nom/Numero")
self.lblch1.grid(column=0,row=0,sticky='w')
self.lblch2.grid(column=1,row=0,sticky='w')
self.lblch3.grid(column=2,row=0,sticky="w")
```

```
    def Apercu(self,fichier,tailleH,tailleV):
        taille = tailleH,tailleV
        posExt = fichier.rfind(".")
        fichierSortie = fichier[:posExt] + ".thumbnail"
        im = Image.open(fichier)
        im.thumbnail(taille)
        im.save(fichierSortie,"JPEG")
```

Comme toujours, le code est disponible sur Pastebin : http://pastebin.com/RM7CgtbT (http://pastebin.com/XtBawJps pour l'original).

Rendez-vous le mois prochain pour étoffer le code.

```
def AfficherAide(self):, def AfficherApropos(self):, def OuvrirBase(self):, def
AfficherMasquerGrille(self):
def ChoixTaillePoints(self,p):, def ChoixTailleAida(self,p):, def Traitement(self):
def CreerPDF(self):, def InfoOriginal(self,fichier):, def RecupererNbCouleurs(self,fichier):
def RecupererHauteurLargeur(self,fichier):, def RecupererHauteurLargeur2(self,fichier):, def
    RecupererCouleurs(self,image):
def Pixeliser(self,im,taillePixel):, def ReduireCouleurs(self,NomImage):
def DessinerLignes(self,im,taillePixel):, def DessinerLignes2(self,im,taillePixel):
def Rgb2Hex(self,rgb):, def RemplirListeDeroulante(self,nomFic):
def TrouverMeilleureDistance(self,r1,g1,b1):
```

Programmer en Python - Partie 56

Depuis quelques mois, nous travaillons sur un générateur de motifs de point de croix. Le mois dernier, nous avons mis en place l'interface utilisateur, maintenant il est temps d'écrire le code qui fait le plus gros du travail. Le mois prochain, nous commencerons à travailler sur la partie qui crée le fichier PDF.

Nous allons travailler d'abord sur les éléments de menu. Le code est ci-dessous.

La variable globale PretPourTraitement est utilisée pour s'assurer que si l'utilisateur appuie sur le bouton de traitement, le système ne va pas chercher à traiter des choses s'il n'y a rien à traiter. Nous utilisons la routine de dialogue askopenfilename intégrée à tkFileDialog pour obtenir le nom du fichier qui contient l'image originale. On obtient alors le nombre de couleurs de l'image originale ainsi que la largeur et la hauteur. Nous sauvons ces valeurs et les affichons dans l'interface graphique. Nous ouvrons ensuite l'image et créons une image miniature à afficher dans la partie gauche du cadre inférieur. La boîte de texte est montrée à droite.

Ensuite, nous écrivons la fonction AfficherMasquerGrille. Elle échange tout simplement deux images dans le label image de droite en se basant sur la variable globale AfficherGrille. Si elle vaut False, nous changeons le texte du bouton Afficher/Masquer, puis définissons la variable AfficherGrille à True et définissons l'image à celle qui contient la grille. Sinon, nous changeons le texte sur le bouton Afficher/Masquer en « Afficher grille », définissons la variable AfficherGrille à False et mettons en place l'image sans grille. Le code se trouve sur la page suivante, en haut à gauche.

La fonction ChoixTaillePoints est déclenchée à chaque fois que la liste déroulante de taille du point est modifiée. Nous récupérons la valeur de la

```
    NomFichierOriginal.set(NomFic)
    NombreCouleursOriginal.set(self.RecupererNbCouleurs(NomFic))
    TailleOriginal.set(self.RecupererHauteurLargeur(NomFic))
    imageMaitresse=Image.open(NomFic)
    imageMaitresse.thumbnail((500,500))
    self.img = ImageTk.PhotoImage(imageMaitresse)
    self.lblImageL['image'] = self.img
    PretPourTraitement = True
```

L'option de menu SauverFichier appellera simplement la routine CreerPDF, quand elle sera finie.

```
    def SauverFichier(self):
        self.CreerPDF()
```

Nous allons bâcler les routines AfficherAide et AfficherAPropos avec une boite de dialogue indiquant que ces options ne sont pas encore disponibles.

```
    def AfficherAide(self):
        tkMessageBox.showinfo(title="Aide",message='Desole,
        la fonction aide est encore inexistante.')

    def AfficherApropos(self):
        tkMessageBox.showinfo(title="About",message='Desole,
        la fonction a propos est encore inexistante.')
```

Nous avons déjà écrit la routine OuvrirBase une douzaine de fois. Vous devez donc savoir ce qu'elle fait.

```
    def OuvrirBase(self):
        global connexion
        global curseur
        #-------------------------------
        connexion = apsw.Connection("floss.db3")
        curseur = connexion.cursor()
```

```
    def RecupererNomFichier(self):
        global PretPourTraitement
        #-------------------------------
        NomFic = tkFileDialog.askopenfilename(parent=racine,filetypes=self.formatsImages ,title="Choisir le fichier a ouvrir...")
```

```
    def AfficherMasquerGrille(self):
        global AfficherGrille
        #--------------------------------
        if AfficherGrille == False:
            self.btnAfficherGrille['text'] = 'Masquer grille'
            AfficherGrille = True
            self.im2=Image.open(self.GrilleImage)
            self.im2.thumbnail((400,400))
            self.img3 = ImageTk.PhotoImage(self.im2)
            self.lblImageR['image'] = self.img3
        else:
            self.btnAfficherGrille['text'] = 'Afficher grille'
            AfficherGrille = False
            self.im2=Image.open(self.ImageTraitee)
            self.im2.thumbnail((400,400))
            self.img3 = ImageTk.PhotoImage(self.im2)
            self.lblImageR['image'] = self.img3
```

liste déroulante et l'affectons à une variable locale.

```
def
ChoixTaillePoints(self,p):

selection =
ComboTaillePoints.get()
```

La fonction ChoixTailleAida (en haut à droite) est très similaire à la fonction ChoixTaillePoints. Nous réglons les variables globales LargeurTissu et HauteurTissu en fonction de la sélection dans la liste déroulante. Nous mettons également à 30×30 par défaut si on choisit 30.

Nous avons une variable appelée PretPourTraitement (ci-dessous) juste au cas où l'utilisateur tente d'exécuter la fonction de traitement avant que l'image ne soit chargée.

Nous pixelisons le fichier original à une matrice de pixels 5×5. Cela nous permet de réduire cette matrice 5×5 à une seule couleur. Nous réduisons ensuite les couleurs, récupérons largeur et hauteur de l'image traitée et réglons la taille pour que l'utilisateur puisse voir quelle sera la taille de l'image résultante.

```
    def ChoixTailleAida(self,p):
        selection = ComboTaille.get()
        if selection != "30":
            pos = selection.find("x")
            largeur = int(selection[:pos])
            hauteur=int(selection[pos+1:])
        else:
            largeur = 30
            hauteur = 30
        LargeurTissu.set(largeur)
        HauteurTissu.set(hauteur)
```

```
# Placer image

self.im2=Image.open(Reduite)

self.im2.thumbnail((500,500))

self.img3 =
ImageTk.PhotoImage(self.im2)

self.lblImageR['image'] =
self.img3

self.ImageTraitee = 'im1.png'
```

Le code ci-dessus met l'image traitée dans l'image qui contiendra l'image traitée. La suite du code créera une grille afin que l'utilisateur ait la grille pour faire le point de croix.

```
self.DessinerLignes(Reduite,5
)

self.DessinerLignes2('output.
png',50)

self.im2 =
Image.open('output2.png')
```

```
    def Traitement(self):
        global PretPourTraitement
        #--------------------------------
        if PretPourTraitement == False:
            tkMessageBox.showinfo(title="ERREUR...",message='Vous devez charger une image originale.')
        else:
            nouvelleImage = self.Pixeliser(NomFichierOriginal.get(),5)
            Reduite = self.ReduireCouleurs(nouvelleImage)
            L,H = self.RecupererHauteurLargeur2(Reduite)
            tail = "{0}x{1}".format(L/5,H/5)
            TailleTraitee.set(tail)
```

```
self.im2.thumbnail((500,500))

self.img3 =
ImageTk.PhotoImage(self.im2)

self.lblImageR['image'] =
self.img3

self.RemplirListeDeroulante('
output.png')

self.GrilleImage =
'output2.png'
```

Nous bâclons la fonction CreerPDF et nous la finirons le mois prochain.

```
    def CreerPDF(self)

tkMessageBox.showinfo(title="
Creer PDF",message='Desole,
la fonction CreerPDF est
encore inexistante.')
```

La routine InfoOriginal() récupère et définit des variables en fonction du format de l'image d'origine, sa taille et son mode.

```
def
OriginalInfo(self,fichier):
  im = Image.open(fichier)
  imFormat = im.format
  imTaille = im.size
  imMode = im.mode

  self.taille = imTaille
  self.imformat = imFormat
  self.immode = imMode
```

La fonction RecupererNbCouleurs utilise la méthode .getcolors pour obtenir le nombre de couleurs dans le fichier image. Nous devons utiliser 1600000 comme paramètre MaxCouleurs parce que, si l'image contient plus de 256 couleurs (ou ce que contient le paramètre), la méthode retourne « None ». Cette fonction est similaire à la fonction RecupererCouleurs sauf que RecupererCouleurs travaille avec une image déjà ouverte. Si vous utilisez RecupererNbCouleurs, vous devez passer un fichier non ouvert.

```
def
RecupererNbCouleurs(self,fichi
er):
  im = Image.open(fichier)
  nbCouleurs =
im.getcolors(1600000)
  self.couleurs =
len(nbCouleurs)
  return self.couleurs
```

Les deux fonctions suivantes renvoient la hauteur et la largeur en pixels du fichier image. La différence entre les deux est que RecupererHauteurLargeur renvoie une chaîne comme 1024×768 et RecupererHauteurLargeur2 renvoie deux nombres entiers.

```
def
RecupererHauteurLargeur(self,
fichier):
  im = Image.open(fichier)
  tmp =
"{0}x{1}".format(im.size[0],i
m.size[1])
  return tmp
```

```
def
RecupererHauteurLargeur2(self
,fichier):
  im = Image.open(fichier)
  return
im.size[0],im.size[1]
```

RecupererCouleurs cherchera le nombre de couleurs dans l'image passée en paramètre. Nous utilisons 1,6 million de couleurs comme paramètre, car la routine image.getcolors renvoie 0 (par défaut) s'il y a plus que 256 couleurs.

```
def
RecupererCouleurs(self,image)
:
  nbCouleurs =
image.getcolors(1600000)
  couleurs = len(nbCouleurs)
```

```
def Pixeliser(self,im,taillePixel):
    image = Image.open(im)
    self.RecupererCouleurs(image)
    image = image.resize((image.size[0]/taillePixel, image.size[1]/taillePixel),Image.NEAREST)
    image = image.resize((image.size[0]*taillePixel, image.size[1]*taillePixel),Image.NEAREST)
    self.RecupererCouleurs(image)
    #image.show()
    image.save('newimage.png')
    return 'newimage.png'
```

```
def ReduireCouleurs(self,NomImage):
    #Reduire couleurs
    nbCouleurs=MaxCouleurs.get()
    image = Image.open(NomImage)
    output = image.convert('P', palette=Image.ADAPTIVE, colors=nbCouleurs)
    x = output.convert("RGB")
    self.RecupererCouleurs(x)
    nbCouleurs = x.getcolors()
    CouleursTraitees.set(len(nbCouleurs))
    x.save('im1.png')
    return 'im1.png'
```

La fonction Pixeliser (page précédente en haut) prend deux paramètres, le nom du fichier image (im) et la taille des pixels que vous voulez. Le travail est effectué par la méthode image.resize. J'ai trouvé cette routine sur le Web à pas mal d'endroits. Dans cet exemple, nous allons passer une taille de pixel de 5, qui fonctionne bien pour des projets de point de croix. Nous disons aussi à la méthode de prendre la couleur du plus proche voisin. Cela renvoie une nouvelle image, que nous enregistrons dans un fichier et retournons le nom de ce fichier.

La routine ReduireCouleurs (page précédente en bas) utilise essentiellement la palette Image.ADAPTIVE afin d'obtenir un nombre très restreint de couleurs.

Il y a deux routines DessinerLignes (en haut à droite). Elles créent la grille dont nous avons parlé plus tôt.

Rgb2Hex() retourne une valeur hexadécimale de la valeur RVB qui est passée. Nous allons l'utiliser pour essayer de comparer les couleurs dans la base de données avec les couleurs de l'image.

```
def Rgb2Hex(self,rgb):
  return '#%02x%02x%02x' % rgb
```

La liste déroulante (ci-dessous) con-

```
def DessinerLignes(self,im,taillePixel):
    global couleurFond1
    #--------------------------------
    image = Image.open(im)
    pixel = image.load()
    for i in range(0,image.size[0],taillePixel):
        for j in range(0,image.size[1],taillePixel):
            for r in range(taillePixel):
                pixel[i+r,j] = couleurFond1
                pixel[i,j+r] = couleurFond1
    image.save('output.png')

def DessinerLignes2(self,im,taillePixel):
    global couleurFond1
    #--------------------------------
    image = Image.open(im)
    pixel = image.load()
    for i in range(0,image.size[0],taillePixel):
        for j in range(0,image.size[1],taillePixel):
            for r in range(taillePixel):
                try:
                    pixel[i+r,j] = couleurFond1
                    pixel[i,j+r] = couleurFond1
                except:
                    pass
    image.save('output2.png')
```

```
def RemplirListeDeroulante(self,nomFic):
    im = Image.open(nomFic)
    nbCouleurs = im.getcolors()
    couleurs = len(nbCouleurs)
    cntr = 1
    for c in nbCouleurs:
        hexcolor = self.rgb2hex(c[1])
        lblCouleur=Label(self.sfFrame,text="                ",bg=hexcolor,relief=GROOVE)
        lblCouleur.grid(row = cntr, column = 0, sticky = 'nsew',padx=10,pady=5)
        pkID = self.TrouverMeilleureDistance(c[1][0],c[1][1],c[1][2])
        sql = "SELECT * FROM DMC WHERE pkID = {0}".format(pkID)
        rset = curseur.execute(sql)
        for r in rset:
            hexcolor2 = r[6]
            dmcnum = r[1]
            nomCouleur = r[2]
        lblCouleur2=Label(self.sfFrame,text="                ",bg="#" + hexcolor2,relief=GROOVE)
        lblCouleur2.grid(row = cntr,column = 1,sticky = 'w',padx=5,pady=5)
        lblCouleur3=Label(self.sfFrame,text = str(dmcnum) + "-" + nomCouleur,justify=LEFT)
        CouleurDMC.set(dmcnum)
        lblCouleur3.grid(row = cntr, column = 2,sticky = "w",padx=1,pady=5)
        cntr += 1
```

tient les couleurs qui seront utilisées pour obtenir les couleurs appropriées de fils. Nous créons simplement des « labels » pour contenir les couleurs (visuelles) et le texte.

Voici la routine (ci-contre) que nous utilisons pour essayer de trouver la meilleure correspondance entre la couleur dans l'image et la couleur dans la base de données. Il existe de nombreux algorithmes différents sur le Web que vous pouvez regarder pour essayer de comprendre leur logique. Cela peut être assez compliqué.

Bon. C'est tout pour ce mois-ci. La prochaine fois, nous allons commencer à créer le fichier de sortie PDF pour que la brodeuse ait un support avec lequel travailler.

Comme toujours, le code est disponible sur Pastebin : http://pastebin.com/d8JUyeKA (http://pastebin.com/DmQ1GeUx pour la version anglaise). Nous continuerons dans les prochains mois. Je dois bientôt me faire opérer et je ne sais pas à partir de quand je pourrai rester assis longtemps. Jusque-là, amusez-vous bien.

```
def TrouverMeilleureDistance(self,r1,g1,b1):
    # dist = math.sqrt(((r1-r2)**2) + ((g1-g2)**2) + ((b1-b2)**2))
    sql = "SELECT * FROM DMC"
    rset = curseur.execute(sql)
    BestDist = 10000.0
    for r in rset:
        pkID = r[0]
        r2 = r[3]
        g2 = r[4]
        b2 = r[5]
        dist = math.sqrt(((r1-r2)**2) + ((g1-g2)**2) + ((b1-b2)**2))
        if dist < BestDist:
            BestDist = dist
            BestpkID = pkID
    return BestpkID
```

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Générateur de modèle de point de croix - Partie 4 - Comprendre pyfPDF

Désolé d'avoir manqué tant de mois. Je ne peux toujours pas rester assis pendant trop longtemps, du coup cet article est plus court que d'habitude. Mon plan initial était de passer directement à la partie du programme qui crée le PDF, mais il y a tellement de choses à comprendre dans cette bibliothèque que j'ai décidé d'utiliser cet épisode comme un tutoriel sur pyfPDF et attaquer la sortie PDF la prochaine fois. Bon, commençons.

FPDF signifie PDF gratuit. Voici un exemple très minimaliste :

```
from fpdf import FPDF

pdf = FPDF()

pdf.add_page()

pdf.set_font('Arial','B',16)

pdf.cell(40,10,'Hello From Python')

pdf.output('example1.pdf','F')
```

La première ligne importe la bibliothèque. La suivante crée une instance de l'objet FPDF. Nous utilisons pour cet exemple les valeurs par défaut, qui sont :

- Portrait
- Unité de mesure = millimètres
- Format = A4

Si vous avez besoin d'utiliser les normes « US », vous pouvez le faire de cette façon :

```
pdf=FPDF('P','in','Letter)
```

Notez que les paramètres sont FPDF (orientation, unités, format) :

- Les valeurs possibles pour l'orientation sont « P » pour portrait et « L » pour paysage (« landscape »).
- Les valeurs possibles pour les unités sont : 'pt' (points), 'mm' (millimètre), 'cm' (centimètre), 'in' (pouces).
- Les valeurs possibles pour le format sont : 'A3', 'A4', 'A5', 'Letter', 'Legal' ou un tuple contenant la largeur et la hauteur exprimées dans l'unité donnée dans le paramètre précédent.

La troisième ligne crée une page pour écrire des données. Remarquez qu'une page n'est pas automatiquement créée lorsque nous créons l'instance de l'objet. L'origine de la page est le coin supérieur gauche et la position de départ se situe par défaut à 1 cm des marges. Les marges peuvent être modifiées avec la fonction SetMargins.

Avant de pouvoir afficher du texte, vous devez appeler pdf.set_font() pour définir une police. Dans la ligne ci-dessus, nous définissons Arial Bold 16 points. Les polices standards valides sont Arial, Times, Courier, Symbol et ZapfDingbats.

Maintenant, nous pouvons imprimer une cellule en appelant pdf.cell(). Une cellule est une zone rectangulaire, éventuellement encadrée, qui contient du texte. L'affichage se fait à la position qui est spécifiée (40,10 cm dans l'exemple ci-dessus). Les paramètres sont :

```
pdf.cell(Width, Height, text, border, line, align, fill, link)
```

où :

- largeur est la largeur de la cellule. Si égale à 0, la largeur va jusqu'à la marge de droite ;
- hauteur est la hauteur de la cellule ;
- texte est la chaîne de texte que vous souhaitez afficher ;
- bordure est soit 0 (pas de bordure, par défaut), 1 pour une bordure, ou une chaîne de tout ou partie des caractères suivants : « L», « T », « B », « R » ;
- ligne indique la position à laquelle on doit aller après l'affichage du texte. Les valeurs sont 0 (vers la droite), 1 (au début de la ligne suivante), 2 (en-dessous). Par défaut c'est 0, et indiquer 1 revient à mettre 0 et appeler ln() immédiatement après ;
- alignement permet de centrer ou aligner le texte dans la cellule. Les valeurs sont « L » (gauche), « C » (centre), « R » (droite) ;
- remplissage définit si le fond est rempli (true) ou transparent (false). Par défaut c'est false.
- Lien est une url ou un identifiant retourné par addlink().

Enfin, le document est fermé et envoyé au fichier avec Output. Les paramètres sont fpdf.output(nom, destination). Si aucun fichier n'est spécifié, la sortie sera envoyé au navigateur. Les options pour la destination sont « I » (en ligne dans le navigateur, par défaut), « F » (fichier local donné

par nom), « D » (envoi au navigateur en forçant un téléchargement de fichier avec le nom passé) et « S » (retourne le document sous forme de chaîne).

Puisque nous allons envoyer nos images de point de croix dans le fichier pdf, nous devrons comprendre la fonction image.

Cette fonction est appelée comme ceci :

```
pdf.image(name,x=None,y=None,
w=0,h=0,type="",link="")
```

Cette fonction place une image. La taille qu'elle occupera sur la page peut être spécifiée de différentes manières :
- la largeur et la hauteur explicite ou
- une dimension explicite.

Les formats supportés sont JPEG, PNG et GIF. Si vous souhaitez utiliser des fichiers GIF, vous devez installer l'extension GD.

Pour les fichiers JPEG, tous les types sont autorisés :
- échelle de gris ;
- couleurs vraies (24 bits) ;
- CMYK (32 bits).

Pour PNG, les types suivants sont acceptés :
- niveaux de gris sur au plus 8 bits (256 niveaux) ;
- couleurs indexées ;
- couleurs vraies (24 bits).

Remarque : l'entrelacement n'est pas permis et, si vous utilisez une version de FPDF antérieure à la 1.7, le canal alpha n'est pas supporté.

J'ai volé cet exemple (à droite) dans le tutoriel de pyFPDF [Ndt : les commentaires ont été francisés].

Vous avez vu assez de choses pour être en mesure d'examiner le programme et comprendre ce qui se passe. Mais dans cet exemple la ligne qui nous intéresse VRAIMENT est la quatrième :

```
this.image('img1.png',10,8,33)
```

Dans ce cas, nous appelons la fonction image avec le nom du fichier, la position x de l'endroit où sera l'image sur la page, la position y et la largeur de l'image.

Maintenant que vous avez une connaissance grossière de la bibliothèque, nous pourrons commencer notre code PDF la prochaine fois.

Jusque-là, passez un bon mois. À bientôt.

```
from fpdf import FPDF

  def header(this):
        # Logo - remplacer par un petit PNG de votre choix
        this.image('img1.png',10,8,33)
        # Arial bold 15
        this.set_font('Arial','B',15)
        # se deplacer vers la droite
        this.cell(80)
        # titre
        this.cell(30,10,'Titre',1,0,'C')
        # saut de ligne
        this.ln(20)
        # instanciation de la classe

pdf=PDF()
pdf.alias_nb_pages()
pdf.add_page()
pdf.set_font('Times','',12)
for i in range(1,41):
    pdf.cell(0,10,'Affichage du numero de ligne
'+str(i),0,1)
pdf.output('example2.pdf','F')
```

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Programmer en Python - Partie 57a

Écrit par Greg D. Walters

Tout d'abord, permettez-moi de remercier tous les lecteurs qui m'ont envoyé des courriels d'espoir et des vœux de prompt rétablissement. C'était super gentil et d'une grande aide. Je tiens également à remercier Ronnie, notre merveilleux rédacteur en chef, pour son soutien et sa patience pendant cette période douloureuse. J'ai encore des problèmes pour rester longtemps assis et du coup je rédige ceci sur plusieurs jours ; j'espère arriver à garder une certaine continuité. Maintenant, place au « spectacle »…

Il n'y a pas très longtemps, j'allais pointer quand le directeur général de mon « travail de jour » m'a appelé dans son bureau. En espérant que c'était juste une conversation de style « comment ça va », je suis entré et me suis assis. Il a alors commencé par : « *J'ai un problème avec mon tableur, et j'espérais que vous pourriez m'aider.* »

Ma vision s'est obscurcie et les trois notes sinistres « Da Da DAAAAAAAAA » des films d'horreur des années 70 et 80 ont sonné dans ma tête, mais plutôt que de courir hors de la salle en hurlant, j'ai demandé innocemment quel était le problème exact. Il a répondu qu'il y avait un souci avec l'une des macros qui « s'arrêtait en plein milieu des calculs ». En revêtant mon chapeau blanc de cow-boy, j'ai dit dans ma meilleure voix de héros : « Ne vous inquiétez pas citoyen. Nous allons remettre ça en état en un rien de temps. » J'ai rapidement découvert que la raison pour laquelle la feuille de calcul plantait sans cérémonie était qu'une cellule dans l'un des 35 classeurs contenait une erreur de division par zéro parce qu'une valeur attendue n'était pas saisie dans une autre cellule dans un autre des 35 classeurs. Permettez-moi de souligner très clairement que ce n'était pas la faute de mon patron. Tout ce qu'il avait demandé était un moyen simple d'obtenir des valeurs représentatives à partir des données. (Les deux phrases précédentes n'ont absolument rien à voir avec le fait que mon patron peut lire cet article ! Ou peut-être que si.)

En retournant à mon bureau, et me débarrassant des faux bits de code accrochés à mon chapeau blanc, j'ai réalisé que ce serait une excellente opportunité de faire un peu d'enseignement. Et nous y sommes. Mais d'abord, nous allons revenir à 1979, lorsque Apple a lancé Visicalc. C'était le premier « système de type formulaire libre de calcul » qui a eu un vrai succès sur le marché. Bien qu'il y ait de nombreux bogues dans le logiciel, le monde a adoré l'idée et les clones (bogues inclus) ont commencé à apparaître sur d'autres systèmes informatiques, comme le Commodore PET et d'autres concurrents d'Apple (y compris Microsoft en 1981 avec un programme appelé Multiplan). Enfin, en 1983, une société appelée Lotus Development Corp. a introduit Lotus 1-2-3. Bien que très proche de Visicalc sur de nombreux aspects, y compris la structure des menus, il était écrit entièrement en langage assembleur x86, ce qui le rendait très rapide, et de nombreux bogues de Visicalc ont été corrigés. Lotus 1-2-3 était si populaire qu'il est devenu une base de référence classique pour tester la « compatibilité PC » d'une machine.

L'avènement des systèmes de formulaires libres de calculs a permis à la personne « normale » de manipuler des nombres d'une manière qui était auparavant du domaine de la programmation. Presque n'importe qui pouvait, en quelques heures, donner un sens à des nombres, créer des tableaux et des graphiques, et partager cette information avec des collègues. Peu de temps après, la capacité d'automatiser certaines parties de la feuille de calcul grâce à des macros et des langages intégrés proches du Basic a donné à ces utilisateurs non-programmeurs encore plus de pouvoir sur leur destin. Ils pouvaient obtenir les réponses eux-mêmes, et également de jolis tableaux et graphiques, sans avoir à faire la queue en attendant l'aide des informaticiens. Cependant, comme nous l'avons tous appris de l'oncle Ben de Peter Parker…

Un grand pouvoir implique de grandes responsabilités

Bientôt la feuille de calcul a été utilisée dans des cas qui relevaient plutôt des bases de données que des feuilles de calcul. Nous avons maintenant des classeurs sur des classeurs qui dépendent d'autres classeurs, et si un petit nombre le long du chemin n'arrive pas à se mettre à jour… eh bien, nous obtenons le vieil effet « château de cartes ».

Je ne pense pas que toutes les feuil-

les de calcul soient mauvaises, mais certaines (lire ici « beaucoup ») auraient dû être converties en bases de données il y a de nombreuses années. Elles sont juste devenu trop grandes et lourdes pour leur propre bien. Si quelqu'un s'était assis avec des programmeurs et avait dit : « *Je vous en prie, aidez-nous* », le monde serait un endroit plus empathique et plus doux.

Maintenant, je descend de ma tribune, et nous arrivons à la véritable raison de l'article de ce mois-ci. Chaque bon programmeur Python devrait avoir parmi ses outils un moyen de traiter avec des feuilles de calcul. Vous ne savez jamais quand vous aurez besoin d'extraire des données d'une feuille de calcul pour les manipuler. Bien qu'il existe plusieurs façons de récupérer des données de feuilles de calcul, comme les fichiers CSV qui ont leurs propres inconvénients, vous avez parfois besoin de lire et d'écrire directement à partir de, et vers, un tableur « actif ». Après avoir cherché, je suis tombé sur une très belle bibliothèque pour accéder à la feuille de calcul problématique de mon patron.

Nous allons ajouter la bibliothèque appelée XLRD, sans doute pour eXceL ReaD (lire Excel). Cette bibliothèque nous permet de lire facilement des données dans des fichiers Excel (.xls, .xlsx et .xlsm) à partir de la version 2.0.

Créons une feuille de calcul Excel pour examiner les fonctionnalités de XLRD. Ouvrez Excel ou OpenOffice ou LibreOffice Calc. Dans la première colonne (A), saisissez les chiffres de 1 à 5 en descendant. Dans la colonne suivante (B), saisissez 6 à 10. Cela devrait ressembler à ceci :

	A	B
1	1	6
2	2	7
3	3	8
4	4	9
5	5	10

Maintenant, sauvegardez la feuille de calcul comme « exemple1.xls » dans le dossier que vous allez utiliser pour enregistrer le code de test. De cette façon, nous n'aurons pas à nous soucier de chemins.

Maintenant téléchargez et installez XLRD : https://pypi.python.org/pypi/xlrd. Nous pouvons l'utiliser comme illustré ci-dessous.

Enregistrez le fichier sous exemple1.py dans le même dossier que la feuille de calcul. Puisque le code est très court, nous allons tout simplement en discuter ici. Bien sûr, la première ligne importe la bibliothèque. Ensuite, nous créons une fonction appelée OuvrirFichier et passons le nom (et le chemin si nécessaire) de la feuille de calcul à la fonction.

Maintenant, nous appelons la méthode open_workbook et récupérons un objet « classeur ». Ensuite, nous utilisons l'attribut nsheets qui retourne le nombre de feuilles actives. Nous pouvons également obtenir le nom des feuilles. Dans ce cas, ce sont ceux par défaut. Nous utilisons la méthode sheet_by_index pour obtenir la Feuille1 dans l'objet premiere_feuille. Maintenant, nous pouvons commencer à récupérer des données. Nous récupérons l'information de la cellule à la position (1,1) qui correspond à la cellule B2 (on compte à partir de 0, donc la cellule A1 serait (0,0)). Nous écrivons les données à partir de là, à la fois ce que contient la cellule et la valeur, pour que nous puissions l'utiliser dans un calcul si l'on veut.

C'était vraiment facile, non ? Maintenant, nous allons faire quelque chose d'un peu plus utile. Entrez le code indiqué sur la page suivante (en haut à droite) et enregistrez-le comme « exemple2.py ». Cet exemple permet d'affi-

```
import xlrd
def OuvrirFichier(chemin):
     # Ouvre et lit un fichier Excel
     classeur = xlrd.open_workbook(chemin)
     # Recupere le nombre de feuilles actives
     print "Nombre de feuilles : ",classeur.nsheets
     # Recupere le nom des ces feuilles
     print "Noms des feuilles : ",classeur.sheet_names()
     premiere_feuille = classeur.sheet_by_index(0)
     cellule = premiere_feuille.cell(1,1)
     print "Cellule en 1,1: ",cellule
     print "Valeur de la cellule en 1,1: ",cellule.value

if __name__ == "__main__":
     chemin = "exemple1.xls"
     OuvrirFichier(chemin)
```

cher le contenu du classeur.

Nous avons déjà utilisé les quatre premières lignes de code dans le premier exemple, nous les laisserons de côté. En utilisant les attributs « sheet.nrows » et « sheet.ncols », on obtient le nombre de lignes et de colonnes. Cela peut être utile, et pas seulement pour savoir à quoi nous avons affaire ; nous pouvons écrire des routines « génériques » qui utilisent ces valeurs dans nos calculs, comme vous le verrez. En fait, nous utilisons « lignes » dans une boucle for pour obtenir les informations de chaque ligne.

Remarquez la ligne qui contient « premiere_feuille.row_slice ». Elle récupère un bloc de cellules d'une ligne donnée. La syntaxe est la suivante :

```
X =
first_sheet.row_slice(RowInQu
estion, Start_Column,
End_Column)
```

Nous avons donc utilisé le nombre de lignes et le nombre de colonnes dans les calculs. La sortie de notre programme devrait ressembler à quelque chose comme çeci…

```
Il y a 5 lignes dans cette
feuille.
Il y a 2 colonnes dans cette
feuille.
[number:1.0, number:6.0]
[number:2.0, number:7.0]
[number:3.0, number:8.0]
```

```
import xlrd
def OuvrirFichier(chemin):
     classeur = xlrd.open_workbook(chemin)
     premiere_feuille = classeur.sheet_by_index(0)
     # recupere le nombre de lignes dans cette feuille
     lignes = premiere_feuille.nrows
     # recupere le nombre de colonnes dans cette feuille
     cols = premiere_feuille.ncols
     print "Il y a %d lignes dans cette feuille." % lignes
     print "Il y a %d colonnes dans cette feuille." % cols
     for l in range(0,lignes):
        cellules = premiere_feuille.row_slice(rowx=l,start_colx=0,end_colx=cols)
        print cellules
if __name__ == "__main__":
     chemin = "exemple1.xls"
     OuvrirFichier(chemin)
```

```
[number:4.0, number:9.0]
[number:5.0, number:10.0]
Appuyez sur une touche pour
continuer…
```

Nous allons voir un exemple de plus avant de terminer cet article. Allez sur la feuille de calcul et placez quelques dates dans la colonne C. Voici à quoi ma feuille de calcul ressemble maintenant (à gauche, colonne 2).

1	6	1/10/2014
2	7	4/15/2015
3	8	6/24/1986
4	9	9/30/1963
5	10	3/3/2000

Vous pouvez utiliser les dates que vous voulez. Maintenant, relancez le programme exemple2.py. Voici la sortie du mien.

```
Il y a 5 lignes dans cette
feuille.
Il y a 3 colonnes dans cette
feuille.
[number:1.0, number:6.0,
xldate:41649.0]
[number:2.0, number:7.0,
xldate:42109.0]
[number:3.0, number:8.0,
xldate:31587.0]
[number:4.0, number:9.0,
xldate:23284.0]
[number:5.0, number:10.0,
xldate:36588.0]
Appuyez sur une touche pour
continuer…
```

Eh bien, ce n'est pas ce que nous attendions. Il semble qu'Excel stocke les dates comme des valeurs qui sont simplement formatées comme nous leur demandons. Ceci peut être utile pour le tri et les calculs, mais, pour afficher les données réelles, cela ne convient pas. Heureusement, les auteurs

```
    for c in cellules:
       if c.ctype == xlrd.XL_CELL_DATE:
          valeur_date = xlrd.xldate_as_tuple(c.value,classeur.datemode)
          dt = str(valeur_date[1]) + "/" + str(valeur_date[2]) + "/" +
 str(valeur_date[0])
          print dt
       else:
          print c.value
```

de la bibliothèque ont déjà pensé à cela. Supprimez la ligne « print cellules » et remplacez-la par le code en bas de la page précédente.

Ici, nous parcourons chaque cellule dans la liste des cellules et vérifions le type de la cellule pour voir si elle est considérée comme un XL_CELL_DATE. Si c'est le cas, alors nous la convertissons en un tuple. Il est stocké sous la forme AAAA, MM, JJ. Nous le rendons alors présentable pour l'afficher sous la forme JJ/MM/AAAA. Voici la sortie de notre nouveau programme…

```
Il y a 5 lignes dans cette
feuille.
Il y a 3 colonnes dans cette
feuille.
1.0
6.0
1/10/2014
2.0
7.0
4/15/2015
3.0
8.0
6/24/1986
4.0
9.0
9/30/1963
5.0
10.0
3/3/2000
Appuyez sur une touche pour
continuer…
```

Juste pour information, les mêmes merveilleuses personnes ont fait une autre bibliothèque appelée XLWT, qui vous permet d'écrire dans des fichiers Excel. Il y a un tutoriel merveilleux et une documentation sur ces deux bibliothèques ici : http://www.python-excel.org/.

Le code source de exemple3.py est sur pastebin : http://pastebin.com/EciU3Fak. (Le code source en anglais se trouve ici : http://pastebin.com/bWz7beBw.)

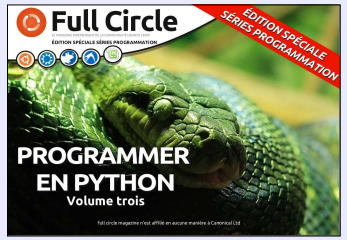
J'espère que je vous verrai le mois prochain.

Greg Walters est propriétaire de Rainy Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Éditions Spéciales Python :

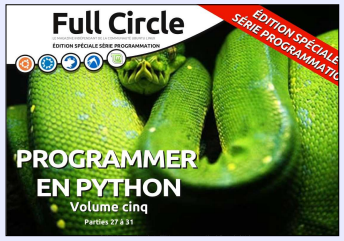
http://www.fullcirclemag.fr/?download/224

http://www.fullcirclemag.fr/?download/230



http://www.fullcirclemag.fr/?download/231

http://www.fullcirclemag.fr/?download/240

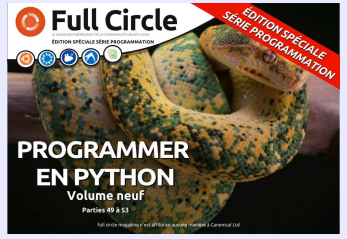


http://www.fullcirclemag.fr/?download/268

http://www.fullcirclemag.fr/?download/272

http://www.fullcirclemag.fr/?download/370

http://www.fullcirclemag.fr/?download/371

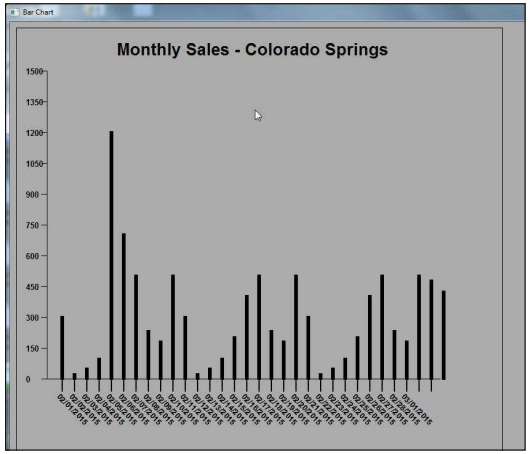


http://www.fullcirclemag.fr/?download/372

Programmer en Python - Partie 58

La dernière fois, nous avons présenté la lecture et l'utilisation de données directement depuis un fichier Excel. Si vous vous souvenez, mon patron (celui de mon travail « de jour ») avait une monstrueuse feuille de calcul dans laquelle, si un calcul plantait, cela mettait fin à tout le processus. Bien, j'ai créé une base de données à partir de cette feuille de calcul dont l'extraction du rapport était facile. Cependant, la feuille de calcul originale créait des jolis tableaux et graphiques que mes patrons aimaient voir. Aussi, j'ai complété le travail pour créer des tableaux afin que chacun soit content…

J'ai passé deux jours à fouiller dans les paquets de tableaux et graphiques déjà existants pour Python, la plupart gratuits et la plupart avec sortie directe en fichier, comme en pdf ou en format graphique (jpg, png, svg). Ce que je cherchais, c'en était un qui sortirait en panneau ou en frame wxPython, de façon à pouvoir l'afficher dans un programme d'interface graphique utilisateur (GUI). J'ai trouvé une solution, mais elle demandait tellement de bibliothèques interdépendantes que la possibilité de la mettre simplement sur une clé USB devenait rapidement nulle.



Comme je suis le genre de type entêté, tenace, qui ne s'avoue jamais vaincu, j'ai décidé d'écrire mon propre programme. Le but original était de faire (au moins) des graphiques à barres et peut-être dans le futur des graphiques linéaires et/ou d'autres types. Il devrait aussi être possible de mettre de la couleur, mais de simples barres noires devraient suffire pour le moment. Il devrait être autonome pour qu'il puisse être appelé comme une bibliothèque. Il n'était pas prévu d'être si générique qu'il deviendrait très compliqué, simplement des dates sur l'axe horizontal (en bas), des valeurs sur l'axe vertical et des barres qui représentent les ventes journalières de la période. De façon à garder le tableau plus ou moins propre, les dates seraient penchées pour éviter d'être écrites les unes sur les autres. Ainsi, ce à quoi je suis arrivé est présenté ici. A gauche, c'est un échantillon de la sortie du code.

Je me répète, rien d'affriolant, pas vraiment sexy, mais il remplit son office. S'il doit devenir plus joli par la suite, je le ferai plus tard.

La première chose que j'ai eu à faire, c'était de récupérer ma documentation sur wxPython pour me rappeler des commandes des graphes. Pour être en mesure de dessiner des graphes, nous utilisons « dc » ou Device Content (contenu de l'élément). C'est une espèce de canevas vierge dans lequel nous pouvons dessiner des lignes, des points et du texte. wxPython offre 9 types différents d'objets dc et j'ai choisi wx.PaintDC qui fonctionne à partir de OnPaintEvent. Nous utiliserons quelques commandes très simples pour dessiner et peindre. Ce sont :

```
dc.DrawLine
dc.SetPen
dc.SetFont
dc.DrawText
dc.DrawRectangle
dc.DrawRotatedText
dc.GetFullTextExtent
```

Ce sont les seules routines wxPython que nous utiliserons, bien qu'il y en ait plein d'autres qui rendraient notre programme beaucoup plus joli. Nous combinerons ces commandes dans nos propres routines « logiques » comme Drawbars, DrawAxis, DrawValues et ainsi de suite. Bien que j'aie pu faire une ou deux grosses routines, je voulais les découper en routines qui aient un sens pour la formation. Allez, commençons à regarder le code. Créez un fichier nommé mongraphe.py. Je n'ai rien trouvé de plus parlant, car PyChart, PyGraph et équivalents sont déjà tous pris. Peut-être que si j'avais eu un peu plus de temps, j'aurai trouvé autre chose, mais ce n'est pas important. Démarrons. D'abord faisons les imports comme nous le faisons toujours.

```
#!/usr/bin/python

# mongraphe.py

import wx

from datetime import date,
datetime, time
```

```
import time
import math
```

Évidemment, nous avons besoin d'importer la bibliothèque wxPython et celle des maths nous aidera pour certains calculs. Les bibliothèques de date et de temps sont utilisées pour les étiquettes de l'axe horizontal.

Quelque chose à garder à l'esprit à partir de maintenant… Quand vous pensez à un dessin dans un contexte, le coin en haut à gauche de la fenêtre conteneur (notre dc) est x=0, y=0. X est l'axe horizontal et Y, l'axe vertical. Plus nous sommes près de l'angle en bas à droite, plus les deux nombres deviennent grands. Dans notre programme, nous commencerons par dessiner une boîte qui définit la zone de notre graphe, qui commence en haut à gauche à X=10, Y=10 et finit à X=800, Y=700. Cependant, avant de passer à cela, nous devons définir une classe pour manipuler les routines et la routine __init__. J'espère que vous vous souvenez des sessions précédentes.

En haut à droite, vous trouvez la définition de classe et la routine __init__.

Notre classe s'appelle Ligne et nous

```
class Line(wx.Fenetre)
def __init__(self, parent, id, TitreFenetre, DonneesEntrantes, TitreGraphe):
      wx.Frame.__init__(self, parent, id, TitreFenetre, size=(1024, 768))
      self.Bind(wx.EVT_PAINT, self.OnPaint)
      self.LargeurBoite = 790
      self.HauteurBoite = 690
      self.TitreGraphe = TitreGraphe
      self.donnees = []
      self.ReglerDonnees(DonneesEntrantes)
      self.Centre()
      self.Show(True)
```

créerons une « wxFrame » pour faire notre dessin. Ce pourrait être un panneau dans une frame ou toute autre option. Mon choix a été d'avoir une fenêtre surgissante dans le graphe avec les données dedans. La première fois que la classe est instanciée, la routine __init__ est appelée avec le nom de l'objet parent, l'identifiant de cet objet, le titre de la fenêtre (dans la barre de titre) les données que l'on veut mettre en graphe et enfin le titre du graphe lui-même. Ensuite, nous créons l'objet wx.frame qui a une taille de 1024×768 pixels. Puis nous relions l'événement paint (qui est appelé quand la frame est créée, déplacée, couverte, découverte…) à notre routine d'événement OnPaint. Souvenez-vous, comme c'est à l'intérieur d'une classe, nous utilisons « self. » pour dire que la routine appartient à la classe et pas à une autre. Nous déclarons les variables (LargeurBoite, HauteurBoite, TitreGraphe, donnees) pour les utiliser plus tard. Après avoir

```
    def DessineBoite(self,dc):
        #Horizontal
        dc.DrawLine(10,10,800,10)
        dc.DrawLine(10,700,800,700)
        #Vertical
        dc.DrawLine(10,10,10,700)
        dc.DrawLine(800,10,800,700)
```

C'est plutôt simple. On passe le dc de la fenêtre puis on dessine 4 lignes. Les paramètres de la fonction DrawLine sont :

```
dc.DrawLine(StartX, StartY, EndX, EndY)
```

défini self.donnees comme une liste vide, nous appelons une routine appelée ReglerDonnees pour trouver l'échelle de nos données, dont nous parlerons plus tard. Enfin, nous déclarons que la fenêtre est centrée sur l'écran et nous appelons la routine Afficher. La routine OnPaint est appelée automatiquement parce que nous créons une Frame.

Ensuite (ci-dessus), nous écrirons une routine qui créera une boîte qui affiche la zone dans laquelle le graphe sera confiné. Ce n'est pas une boîte découpante ou contraignante, c'est simplement pour attirer l'œil de l'utilisateur sur ce que nous voulons qu'il regarde.

Pas vraiment difficile. Nous utiliserons la fonction Drawline plusieurs

```
def DessineAxe(self,dc):
    # Horizontal
    dc.DrawLine(60,580,700,580)
    # Vertical
    dc.DrawLine(60,580,60,80)
```

fois tout au long de ce programme. Ensuite, nous créerons une routine qui tracera les lignes d'axes X (horizontal) et Y (vertical) à l'écran. Nous passons à nouveau le dc de la fenêtre à la routine.

Pour ce qui est de la méthode DrawLine dont nous venons de parler, il n'y a rien d'extraordinaire. Nous dessinons une ligne de 580 pixels qui descend le long du Frame, commençant à X=60 et terminant à X=700. Ensuite nous traçons une ligne qui part de X=60, Y=580 et va jusqu'à X=60, Y=80. Cette ligne est tirée de bas en haut, mais vous pourriez la tracer de haut en bas.

Ensuite, nous nous occuperons de la routine DessineTitre. Une fois encore, nous passons le dc de la fenêtre ainsi que le texte que nous voulons dessiner. Durant le processus, pensez que l'on dessine du texte plutôt que l'afficher. Ce n'est pas grand chose, mais ça aide.

Cette routine est plus longue que la plupart des autres, mais c'est dû en partie aux commentaires que j'ai mis. Les deux premières lignes initialisent la police et le style d'écriture que nous utiliserons. Dans la première ligne (SetFont), nous définissons la police qui sera celle par défaut, 20 points, pas italique et grasse. Ensuite, nous déclarons noire la couleur du crayon et la largeur à 20. Maintenant nous devons estimer la largeur du texte pour le centrer dans la boîte. Nous obtenons cette information en appelant GetFullTextExtent avec le texte que nous voulons dessiner, en donnant la police et sa taille, la largeur du trait et tout ce que nous venons de définir. Le tuple qui est retourné contient Width, Height, Decent (largeur, hauteur, décalage - jusqu'à quel point des lettres comme « g » ou « y » passeront sous la ligne de base) et toute espace initiale. Pour nos besoins, seule la largeur nous importe. Si vous vous souvenez, nous avons défini une largeur de boîte de 790 dans la fonction __init__. Pour trouver le centre de notre texte dans la boîte, nous prenons la largeur de la boîte moins la largeur du texte et nous divisons par 2. Ce sera la valeur X à utiliser pour tracer le texte. Enfin, nous réinitialisons la taille du crayon et la couleur. Plutôt que d'utiliser des valeurs par défaut prises on ne sait où, nous aurions pu appeler la fonction dc.GetPen avant de commencer, mais quand j'ai commencé le projet, je n'y ai pas pensé.

Notre prochaine routine dessinera les traits d'échelle le long de l'axe horizontal en bas du graphe. Nous les

```
def DessineTitre(self,dc,txt):
        dc.SetFont(wx.Font(20,wx.DEFAULT,wx.NORMAL,wx.BOLD))
        dc.SetPen(wx.Pen(wx.NamedColour('black'),20))
        # Recupere la longueur du texte a dessiner
        vals = dc.GetFullTextExtent(txt)
        # Retourne
(Largeur,hauteur,Decalage,espacementInitial)
        # Recupere la position gauche (x) pour centrer le texte
        txtleft = (self.LargeurBoite-vals[0])/2
        dc.DrawText(txt,txtleft,30)
        # Raz taille et couleur du stylo
        dc.SetPen(wx.Pen(wx.NamedColour('black'),2))
```

```
    def DessineBarresDates(self,dc,dcount):
        for cntr in range(1,dcount+1):
            dc.DrawLine(65+(cntr*20),580,65+(cntr*20),600)
```

voulons équidistants tout le long de la ligne. Nous passons (comme d'habitude) dc et une valeur que j'ai appelé dcount qui est le nombre de dates que nous voulons afficher. Comme le nombre de jours d'un mois varie entre 28 et 31, j'ai voulu que ce soit un peu dynamique. Nous utilisons simplement une boucle for pour compter le nombre de lignes à tracer, lesquelles tracer et où. Si vous avez été très attentifs, nous démarrerons les lignes à la position 85, elles auront 20 pixels de haut et seront espacées de 20 pixels.

Quand nous passons au tracé des dates sur le graphe, nous voulons les dessiner en biais. De cette manière, les textes ne se chevaucheront pas et, avouons-le, ce sera plus chouette. Pour cela, nous utiliserons la fonction DessineTexteRot. La fonction prend le texte que nous voulons voir dessiné, la position en X et Y comme point de départ et l'angle que nous choisissons pour le tracé. Dans le cas présent, nous voulons un texte tourné de 45 degrés en rotation anti-horaire, ce qui s'écrit « -45 ». Nous réglerons les paramètres de la police et du crayon à chaque tracé du texte. Nous parlerons de la véritable fonction de dessin de date un peu plus tard.

```
def DessineTexteRot(self,dc,txt,x,y):

dc.SetFont(wx.Font(10,wx.DEFAULT,wx.NORMAL,wx.BOLD))
      dc.SetPen(wx.Pen(wx.NamedColour('black'),20))
      dc.DrawRotatedText(txt,x,y,-45)
```

Nous voudrons aussi tracer les valeurs le long de l'axe vertical, avec des traits d'échelle tout le long. Si nous avions chaque fois la même étendue des données, ce serait facile à faire. Cependant, la réalité montre que la plage des données de notre graphe peut varier d'un mois sur l'autre. Une fois, la valeur la plus haute peut être 300. La fois suivante, cela pourrait être 3 000. Comment créer une routine générique qui en tient compte ? Je vais essayer ici de vous expliquer mon raisonnement.

Vous pourriez vous demander pourquoi j'ai choisi une valeur de 500 pour l'axe vertical si je trace une ligne de 80 à 580 (en réalité de 580 à 80). J'ai choisi d'utiliser une « profondeur visuelle » de 500 pixels pour contenir les valeurs. De cette façon, vous pouvez créer un facteur d'échelle basé sur un module de 500.

Disons que pour un calcul donné notre valeur maximum sera 395. Nous pourrions simplement tracer une barre de 395 pixels de haut pour représenter la valeur. Au calcul suivant, ce maximum est de 2 345. Si nous essayons de tracer la barre à sa pleine hauteur, ça dépassera le haut du graphe. De façon à montrer cette valeur, je dois l'arrondir au 500 le plus près au-dessus, c'est-à-dire 2 500, à prendre comme valeur la plus haute de l'axe. Nous pouvons alors mettre à l'échelle en divisant 2 500 par 500 soit un facteur d'échelle de 5. Maintenant, si nous prenons nos données et que nous divisons chacune par le facteur d'échelle, nous pouvons tracer les valeurs, qui tiendront dans le graphe.

Aussi (voir en haut à droite), nous avons besoin de trouver la valeur la plus haute dans nos données et de l'arrondir au multiple de 500 supérieur le plus proche. Ainsi, pour 375, ce sera 500 ; pour 3 750, ce sera 4 000 et ainsi de suite.

Ensuite, nous devons décider quel type de données nous allons utiliser. Nous verrons plus loin dans le programme que je fournis deux types différents de données dans les listes. L'un assure que les plages de dates que nous utiliserons, le long de l'axe des X, sont les données pour octobre, mais vous pouvez facilement suivre le code (montré dans un petit instant) et changer pour le mois que vous voulez. La seconde liste de données est plus générique et fournit à la fois une date et une valeur comme une liste de tuples. Ceci permet de passer des données de n'importe quelle période. La date est une chaîne et la valeur est soit un entier, soit en virgule flottante. La fonction Regler-

```
    #==================================
    # Arrondi au 500 le plus proche
    #==================================
    def arrondi(self,x):
        return int(math.ceil(x/500.0))*500
```

```
  def ReglerDonnees(self,DonneesAUtiliser):
      if type(DonneesAUtiliser[1]) is tuple:
          self.ListeDates=[]
          self.ListeValeurs=[]
          for l in DonneesAUtiliser:
              self.ListeDates.append(l[0])
              self.ListeValeurs.append(l[1])
          self.ValeurMax =
self.arrondi(max(self.ListeValeurs))
          self.ValeurEchelle = self.ValeurMax/500
      else:
          self.ListeValeurs=[]
          self.ListeDates=[]
          for l in DonneesAUtiliser:
              self.ListeValeurs.append(l)
          self.ValeurMax =
self.arrondi(max(self.ListeValeurs))
          self.ValeurEchelle = self.ValeurMax/500
```

Donnees regarde la première valeur de la liste de données et détermine si c'est un tuple. Si c'est le cas, nous supposons que la structure de la liste correspond à la seconde option, sinon, c'est la première.

Si c'est un tuple, nous créons deux listes, une pour les dates et une pour les valeurs. Ensuite, nous parcourons la liste en la séparant en deux listes. Une fois cela fait, nous trouvons la plus haute valeur (max(Self.ListeValeurs)) et nous lançons la fonction d'arrondi (voir ci-dessus) pour déterminer notre facteur d'échelle. Si les

```
def DessineValeurs(self,dc):
    c2 = 0
    for cntr in range(580,30,-50):
        dc.SetPen(wx.Pen(wx.NamedColour('black'),1))
        dc.DrawLine(60,cntr,50,cntr)
        dc.SetFont(wx.Font(10,wx.DEFAULT,wx.NORMAL,wx.BOLD))
        dc.SetPen(wx.Pen(wx.NamedColour('black'),20))

        dc.DrawText(str(c2),26,cntr-7)
        c2 = c2 + (50 * self.ValeurEchelle)
```

données ne sont pas en tuples, nous effaçons les DEUX listes et faisons les mêmes étapes qu'au-dessus.

Maintenant que nous avons notre facteur d'échelle, nous pouvons tracer les traits d'échelle et les valeurs qui vont représenter l'axe vertical. Nous utilisons à nouveau une boucle for, cette fois-ci de 580 à 30 par pas de -50 le long de la ligne, en traçant des traits de 10 pixels. Après, nous configurons la police (juste au cas où elle aurait changé) et nous dessinons chaque valeur.

Maintenant, regardons les routines qui créeront les traits d'échelle pour les dates le long de l'axe des X si nous choisissons d'avoir une simple liste de données sans inclure les dates. Nous avons deux routines de renfort, une appelée DateToStamp et l'autre Timestamp2Date (Oui, j'étais un peu fainéant quand j'ai écrit celle-ci.) Plutôt que de passer par un paquet de routines DateTime compliquées pour déterminer le nombre de jours d'un mois donné, je vais utiliser une date de début et une date de fin et convertir les deux en horodatage Unix pour obtenir le bon jour du mois dans la séquence. Je vous ai montré la routine DateToStamp précédemment et la routine Timestamp2Date exécute le processus inverse.

```
    def DessineBarres(self,dc):
        dc.SetPen(wx.Pen(wx.NamedColour('black'),5))
        for cntr in range(0,len(self.ValList)):
            dc.DrawRectangle(84 + (cntr* 20),580,2,self.ValList[cntr]/-self.ScaleValue)
```

```
    #==================================
    #  Convertit dd/mm/yy en timestamp unix
    #==================================
    def DateToStamp(self,x):
        x = x+" 00:00:00"
        return(time.mktime(time.strptime(x, "%d/%m/%Y %H:%M:%S")))
    #==================================
    #  Convertit un horodatage unix en dd/mm/yy
    #==================================
    def Timestamp2Date(self,tstmp):
        return datetime.fromtimestamp(int(tstmp)).strftime('%d/%m/%Y')
```

La routine suivante prend les dates de début et de fin, comme présenté auparavant, les convertit en horodatage Unix, puis ajoute 86 400 (le nombre de traits dans une période de 24 heures) pour être sûr d'avoir la dernière valeur de la séquence, puis utilise une autre boucle for pour dessiner le texte en biais où nous le voulons.

Nous arrivons maintenant au gestionnaire d'événements OnPaint qui appelle toute les routines utilitaires que nous devons gérer. Souvenez-vous, en utilisant la routine PaintDC, à chaque fois que la fenêtre est bougée, redimensionnée, couverte ou découverte, le gestionnaire d'événement OnPaint est appelé, assurant de ce fait que notre graphe sera persistant.

```
    #==================================
    #  Dessine les dates en biais
    #==================================
    def DessineDates(self,dc,startdate,enddate):
        sd = int(self.DateToStamp(startdate))
        ed = int(self.DateToStamp(enddate))
        ed = ed + 86400
        stp = 1
        for cntr in range(sd,ed,86400):
            dt = self.Timestamp2Date(cntr)
            self.DessineTexteRot(dc,dt,65+(stp*20),600)
            stp = stp + 1
```

D'abord (voir à gauche au milieu de la page suivante), nous obtenons une instance de notre dc, puis nous appelons les routines DessineBoite, DessineAxe, DessineTitre et DessineBarresDates. Ensuite, nous déterminons si la ListeDates (créée dans la routine ReglerDonnees appelée par la routine __init__) est vide ou si des dates peuvent en être extraites. Si c'est le cas, nous appelons la routine DessineDates avec les bonnes valeurs. Puis nous appelons la routine DessineValeurs et, enfin, la routine DessineBarres. Maintenant, vous devriez comprendre pourquoi j'ai découpé le sujet en tout petits bouts.

La dernière chose que nous avons à regarder est la routine d'exécution.

Vous vous souvenez probablement que le « if __name__ == "__main__" » fonctionne si nous appelons le programme seul plutôt que comme une bibliothèque. Les deux lignes suivantes sont les données fictives que j'ai utilisées pour tester le programme. Vous pouvez commenter la première et lancer le programme avec la seconde ligne qui est celle qui utilise le tuple. Les trois dernières lignes instancieront les routines wxPython, puis la classe Ligne et enfin appellera la routine wxPython app.MainLoop pour lancer la fenêtre.

Et voilà notre programme et notre bibliothèque personnalisés de graphe/tableau. J'ai mis le code complet sur pastebin à : http://pastebin.com/m2feeh5P.

Le code en français se trouve à : http://pastebin.com/fJ00bhud.

Jusqu'à la prochaine fois, amusez-vous bien à coder.

```
    #==================================
    #  Routine principale
    #==================================
    def OnPaint(self,event):
        dc = wx.PaintDC(self)
        self.DessineBoite(dc)
        self.DessineAxe(dc)
        self.DessineTitre(dc,self.TitreGraphe)
        # Barres de dates et dates
        self.DessineBarresDates(dc,31)
        leng = len(self.ListeDates)
        if leng > 0:
            sd = self.ListeDates[0]
            ed = self.ListeDates[4]
            self.DessineDates(dc,sd,ed)
        else:
            self.DessineDates(dc,"02/01/2015","03/01/2015")
        # Barres de valeurs - Dessine 10 barres
        self.DessineValeurs(dc)
        # Enfin on dessine les barres de donnees
        self.DessineBarres(dc)
```

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

FCM N°100 ENQUÊTE

La question est :

Quelles sont vos saveurs et versions préférées/détestées ?

Répondez à ce sondage rapide et nous publierons les résultats dans le FCM n° 100

http://goo.gl/DPt2q0

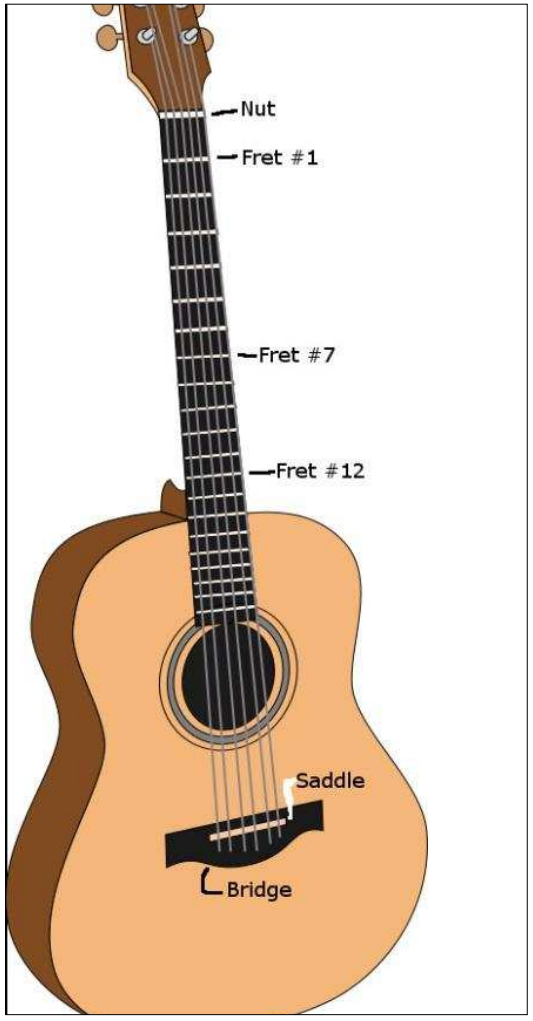
```
if __name__ == "__main__":
    donnees =
(300,20,47,96,1200,700,500,230,179,500,300,20,47,96,200,400,500,230,179,500,300,20,47,96,200,400,500,230,179,500,475,423)
    #donnees =
(("02/01/2015",169.63),("02/02/2015",188.81),("02/03/2015",61.85),("02/04/2015",94.53),("02/05/2015",235.85))
    app = wx.App()
    Ligne(None, -1, 'Bar Chart',donnees,"Ventes mensuelles - Colorado Springs")
    app.MainLoop()
```

Programmer en Python - Partie 59

Tout d'abord, permettez-moi de fêter un joyeux numéro 100 à Ronnie et l'équipe. C'est un privilège de faire partie de ce numéro.

Cette fois-ci, j'ai pensé partager certaines informations sur ma nouvelle obsession. J'ai commencé à réparer et à construire des instruments de musique à cordes comme les guitares et les violons. Croyez-le ou non, il y a pas mal de maths dans les instruments de musique. Aujourd'hui, nous allons examiner certaines des mathématiques en rapport avec la longueur des cordes et l'emplacement des frettes sur le manche.

Jetez un œil à l'image de la guitare. J'ai annoté divers éléments dans l'image. Parmi les choses importantes, le sillet de tête vers le haut du manche, les frettes, le chevalet près du bas, et la « ligne » blanche près du chevalet appelée le sillet de chevalet. Le but des frettes est de créer un endroit parfait pour modifier la longueur de la corde afin de créer une note juste. Les positions de ces frettes ne sont pas arbitraires, mais mathématiquement déterminées.



En effet, la physique des cordes vibrantes nous dit que prendre la moitié de la longueur de corde vibrante d'une corde théoriquement parfaite double la fréquence des vibrations. Dans le cas d'une guitare, cette longueur de corde se situe entre le sillet de tête et le sillet de chevalet. Cette distance est appelée le diapason de la guitare. La demi-longueur qui permet de doubler la fréquence est la frette n° 12. Si c'est fait correctement, placer légèrement votre doigt sur la corde à cet endroit vous donne une note agréable. Il y a quelques autres positions où cela se produira, mais la 12e frette devrait être l'endroit parfait pour ce doublement, montant la note d'une octave.

Différents diapasons vont créer des tonalités et des résultats différents. Par exemple, les guitares Fender Stratocasters® ont un diapason de 25 ½", ce qui produit un son de cloche riche et fort. En revanche, les guitares Gibson utilisent souvent un diapason de 24 ¾". Cela crée une tension de corde inférieure, facilitant la sensation de jeu et un ton plus chaud. D'autres fabricants de guitares ont décidé qu'un diapason de 25" permet un son plus clair que les deux diapasons « standards » précédents.

Ainsi, avec la capacité d'un fabricant de guitares à proposer son propre diapason, l'espacement des frettes devra être recalculé. Les luthiers (fabricants de guitares) gèrent cela depuis des centaines d'années.

Par le passé, il y avait une technique appelée la règle des 18, qui consistait à diviser successivement par 18 le diapason moins le décalage de la frette précédente. En procédant ainsi, les sons étaient de plus en plus faibles au fur et à mesure qu'on allait vers les aigus. De nos jours, on utilise une constante différente. Cette constante est 17,817. En utilisant cette « nouvelle » constante, la 12e frette ou octave est positionnée exactement à la moitié de la longueur de la corde.

Ces calculs sont assez faciles à faire avec un papier et un crayon ou une simple calculatrice, mais il est tout aussi facile de créer un programme Python pour calculer à notre place en une seconde. Une fois que vous avez les positions, vous sciez simplement une fente pour la frette aux positions correctes et ensuite insérez les frettes au marteau.

Alors, jetons un coup d'œil au programme.

Nous voulons créer un programme qui demande le diapason de la guitare (ou de la basse), fait les calculs et ensuite affiche les distances. Les calculs et toutes les longueurs sont tous retournés en pouces, aussi, pour tous nos amis qui utilisent le système métrique, veuillez ajouter les conversions appropriées. Après presque 5 ans, vous devriez être capable de faire cela facilement.

On n'a pas besoin d'importer des bibliothèques, donc nous allons commencer par la définition de deux variables.

```
Diapason = 0

LongueurCumulee = 0
```

Ensuite, nous allons créer une routine (en haut à droite) qui sera appelée à plusieurs reprises au fur et à mesure que nous « avançons vers le bas » du manche. Nous passerons deux valeurs à cette routine : le diapason et la distance cumulée du sillet de tête à la frette précédente.

Dans cette routine, on prend le diapason, on soustrait la distance cumulée et on attribue cette valeur à ChevaletAFrette. Nous prenons ensuite cette valeur, divisons par notre constante (17,817), ajoutons à la distance cumulée et retournons cette valeur à notre routine d'appel. Rappelez-vous, nous aurions pu simplement retourner la valeur calculée sans l'assigner à une variable. Toutefois, si jamais nous voulons vérifier les valeurs calculées, c'est plus facile à faire si nous assignons la valeur avant de la renvoyer.

Maintenant, nous allons écrire notre routine de travail. Nous avons fait ce genre de chose à plusieurs reprises dans le passé. Nous allons lui passer le diapason et elle va boucler jusqu'à 24 frettes (range(1,25)). Même si votre projet a moins de 24 frettes, vous aurez les positions correctes de toutes vos frettes. Je choisis 24 parce que c'est le maximum de frettes pour la plupart des guitares. Lorsque nous entrons dans la boucle, nous vérifions le nombre de frettes (x) et s'il vaut 1, nous passons la longueur cumulée à 0, car cela est le premier calcul. Sinon, nous passons la dernière longueur cumulée qui devient le résultat de la routine de calcul. Enfin, nous affichons chaque numéro de frette suivi par une version formatée de la longueur

```
def CalculerEspacement(Diapason, STAF):
        ChevaletAFrette = Diapason - STAF
        SilletTeteAFrette = (ChevaletAFrette/17.817) + STAF
        return SilletTeteAFrette
```

cumulée.

Enfin, nous avons le code qui demande le diapason. Je suis sûr que vous vous souvenez du format de la routine raw_input, puisque nous l'avons déjà utilisé de nombreuses fois. Mais peut-être avez-vous oublié ceci : raw_input renvoie toujours une chaîne, donc lorsque nous envoyons la saisie à la routine FaireTravail, nous devons la convertir en nombre à virgule flottante pour que la routine fonctionne correctement. Bien sûr, nous pourrions simplement la passer comme une chaîne, mais nous aurions à faire la conversion dans la routine FaireTravail.

```
Diapason = raw_input("Entrez
le diapason de la guitare   ")

FaireTravail(float(Diapason))
```

Vous pourriez vous demander à quoi sert ce programme si vous ne construisez pas une guitare à partir de zéro. Il peut être utile lorsque vous cherchez à acheter une guitare d'occasion ou à essayer de régler une guitare avec un chevalet flottant. Également, si vous êtes un joueur de guitare, c'est peut-être quelque chose que vous ne saviez pas sur les guitares.

Bien sûr, le code en français est disponible sur pastebin : http://pastebin.com/br6tHAUS (le code original se trouve à : http://pastebin.com/A2RNECt5).

```
def FaireTravail(Diapason):
        LongueurCumulee = 0
        for x in range(1,25):
                NumeroFrette = x
                if NumeroFrette == 1:
                        LongueurCumulee = CalculerEspacement(Diapason,0)
                else:
                        LongueurCumulee = CalculerEspacement(Diapason,LongueurCumulee)
                print("Frette=%d,SilletTeteAFrette=%.3f" %
(NumeroFrette,LongueurCumulee))
```

Écrire pour le Full Circle Magazine

Lignes directrices

Notre seule règle : tout article **doit avoir un quelconque rapport avec Ubuntu ou avec l'une de ses dérivées (Kubuntu, Xubuntu, Lubuntu, etc.).**

Autres règles

• Les articles ne sont pas limités en mots, mais il faut savoir que de longs articles peuvent paraître comme série dans plusieurs numéros.

• Pour des conseils, veuillez vous référer au guide officiel *Official Full Circle Style Guide* ici : http://url.fullcirclemagazine.org/75d471

• Utilisez n'importe quel logiciel de traitement de texte pour écrire votre article – je recommande LibreOffice –, mais le plus important est d'en **VÉRIFIER L'ORTHOGRAPHE ET LA GRAMMAIRE !**

• Dans l'article veuillez nous faire savoir l'emplacement souhaité pour une image spécifique en indiquant le nom de l'image dans un nouveau paragraphe ou en l'intégrant dans le document ODT (OpenOffice/LibreOffice).

• Les images doivent être en format JPG, de 800 pixels de large au maximum et d'un niveau de compression réduit.

• Ne pas utiliser des tableaux ou toute sorte de formatage en **gras** ou *italique*.

Lorsque vous êtes prêt à présenter l'article, envoyez-le par courriel à : articles@fullcirclemagazine.org.

Si vous écrivez une critique, veuillez suivre ces lignes directrices :

Traductions

Si vous aimeriez traduire le Full Circle dans votre langue maternelle, veuillez envoyer un courriel à ronnie@fullcirclemagazine.org et soit nous vous mettrons en contact avec une équipe existante, soit nous pourrons vous donner accès au texte brut que vous pourrez traduire. Lorsque vous aurez terminé un PDF, vous pourrez télécharger votre fichier vers le site principal du Full Circle.

Auteurs francophones

Si votre langue maternelle n'est pas l'anglais, mais le français, ne vous inquiétez pas. Bien que les articles soient encore trop longs et difficiles pour nous, l'équipe de traduction du FCM-fr vous propose de traduire vos « Questions » ou « Courriers » de la langue de Molière à celle de Shakespeare et de vous les renvoyer. Libre à vous de la/les faire parvenir à l'adresse mail *ad hoc* du Full Circle en « v.o. ». Si l'idée de participer à cette nouvelle expérience vous tente, envoyez votre question ou votre courriel à :

webmaster@fullcirclemag.fr

Écrire pour le FCM français

Si vous souhaitez contribuer au FCM, mais que vous ne pouvez pas écrire en anglais, faites-nous parvenir vos articles, ils seront publiés en français dans l'édition française du FCM.

CRITIQUES

Jeux/Applications

Si vous faites une critique de jeux ou d'applications, veuillez noter de façon claire :

• le titre du jeu ;
• qui l'a créé ;
• s'il est en téléchargement gratuit ou payant ;
• où l'obtenir (donner l'URL du téléchargement ou du site) ;
• s'il est natif sous Linux ou s'il utilise Wine ;
• une note sur cinq ;
• un résumé avec les bons et les mauvais points.

Matériel

Si vous faites une critique du matériel veuillez noter de façon claire :

• constructeur et modèle ;
• dans quelle catégorie vous le mettriez ;
• les quelques problèmes techniques éventuels que vous auriez rencontrés à l'utilisation ;
• s'il est facile de le faire fonctionner sous Linux ;
• si des pilotes Windows ont été nécessaires ;
• une note sur cinq ;
• un résumé avec les bons et les mauvais points.

Pas besoin d'être un expert pour écrire un article ; écrivez au sujet des jeux, des applications et du matériel que vous utilisez tous les jours.

MÉCÈNES

MÉCÈNES MENSUELS

2016 :
Bill Berninghausen
Jack McMahon
Linda P
Remke Schuurmans
Norman Phillips
Tom Rausner
Charles Battersby
Tom Bell
Oscar Rivera
Alex Crabtree
Ray Spain
Richard Underwood
Charles Anderson
Ricardo Coalla
Chris Giltnane
William von Hagen
Mark Shuttleworth
Juan Ortiz
Joe Gulizia
Kevin Raulins
Doug Bruce
Pekka Niemi
Rob Fitzgerald
Brian M Murray
Roy Milner
Brian Bogdan
Scott Mack
Dennis Mack
John Helmers
JT
Elizabeth K. Joseph
Vincent Jobard
Chris Giltnane
Joao Cantinho Lopes
John Andrews

2017 :

DONS UNIQUES

2016 :
John Niendorf
Daniel Witzel
Douglas Brown
Donald Altman
Patrick Scango
Tony Wood
Paul Miller
Colin McCubbin
Randy Brinson
John Fromm
Graham Driver
Chris Burmajster
Steven McKee
Manuel Rey Garcia
Alejandro Carmona Ligeon
siniša vidović
Glenn Heaton
Louis W Adams Jr
Raul Thomas
Pascal Lemaitre
PONG Wai Hing
Denis Millar
Elio Crivello
Rene Hogan
Kevin Potter
Marcos Alvarez Costales
Raymond Mccarthy
Max Catterwell
Frank Dinger
Paul Weed
Jaideep Tibrewala
Patrick Martindale
Antonino Ruggiero
Andrew Taylor

2017 :
Linda Prinsen
Shashank Sharma

patreon

CHA CHA CHA CHANGEMENT

Notre administrateur est parti, pour de nombreux mois, sans rien dire à personne et je ne savais pas du tout, ni si, ni quand, les frais du site seraient ou ne seraient pas payés. Au départ, nous devions déménager le nom de domaine et le site, qui aurait été hébergé chez moi, et, finalement, j'ai réussi à retrouver l'admin et à me faire transférer le nom de domaine ainsi que l'hébergement du site.

Le nouveau site fonctionne dès à présent. D'ÉNORMES remerciements à Lucas Westermann (Monsieur Command & Conquer) d'avoir bien voulu prendre du temps sur ses loisirs pour recréer complètement le site, ainsi que les scripts, à partir de zéro.

J'ai fait la page Patreon pour pouvoir recevoir de l'aide financière pour ce qui concerne le domaine et les frais d'hébergement. L'objectif annuel a été atteint rapidement grâce à ceux dont les noms figurent sur cette page. Pas d'inquiétude à avoir : le FCM ne va pas disparaître. Plusieurs personnes ont demandé une option PayPal (pour un don ponctuel) et j'ai donc rajouté un bouton sur le côté du site.

Merci infiniment à tous ceux qui ont utilisé Patreon et le bouton PayPal. Cela m'a beaucoup aidé.

https://www.patreon.com/fullcirclemagazine

COMMENT CONTRIBUER

FULL CIRCLE A BESOIN DE VOUS !

Un magazine n'en est pas un sans articles et Full Circle n'échappe pas à cette règle. Nous avons besoin de vos opinions, de vos bureaux et de vos histoires. Nous avons aussi besoin de critiques (jeux, applications et matériels), de tutoriels (sur K/X/L/Ubuntu), de tout ce que vous pourriez vouloir communiquer aux autres utilisateurs de *buntu. Envoyez vos articles à :

articles@fullcirclemagazine.org

Nous sommes constamment à la recherche de nouveaux articles pour le Full Circle. Pour de l'aide et des conseils, veuillez consulter l'Official Full Circle Style Guide :
http://url.fullcirclemagazine.org/75d471

Envoyez vos **remarques** ou vos **expériences** sous Linux à : letters@fullcirclemagazine.org

Les tests de **matériels/logiciels** doivent être envoyés à : reviews@fullcirclemagazine.org

Envoyez vos **questions** pour la rubrique Q&R à : questions@fullcirclemagazine.org

et les **captures d'écran** pour « Mon bureau » à : misc@fullcirclemagazine.org

Si vous avez des questions, visitez notre forum : fullcirclemagazine.org

FCM n° 119

Date de parution du numéro en langue anglaise :
Vendredi 31 mars 2017.

Équipe Full Circle

Rédacteur en chef - Ronnie Tucker
ronnie@fullcirclemagazine.org

Webmaster - Lucas Westermann
admin@fullcirclemagazine.org

Correction et Relecture
Mike Kennedy, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred, Jim Dyer et Emily Gonyer

Remerciements à Canonical, aux nombreuses équipes de traduction dans le monde entier et à **Thorsten Wilms** pour le logo du FCM.

Pour la traduction française :
http://www.fullcirclemag.fr

Pour nous envoyer vos articles en français pour l'édition française :
webmaster@fullcirclemag.fr

Pour les Actus hebdomadaires du Full Circle :

Vous pouvez vous tenir au courant des Actus hebdomadaires en utilisant le flux RSS : http://fullcirclemagazine.org/feed/podcast

Ou, si vous êtes souvent en déplacement, vous pouvez obtenir les Actus hebdomadaires sur Stitcher Radio (Android/iOS/web) : http: www.stitcher.com/s?fid=85347&refid=stpr

et sur TuneIn à : http://tunein.com/radio/Full-Circle-Weekly-News-p855064/

Obtenir le Full Circle en français :

http://www.fullcirclemag.fr/?pages/Numéros

Obtenir le Full Circle Magazine :

Format EPUB - Les éditions récentes du Full Circle comportent un lien vers le fichier epub sur la page de téléchargements. Si vous avez des problèmes, vous pouvez envoyer un courriel à : mobile@fullcirclemagazine.org

Issuu - Vous avez la possibilité de lire le Full Circle en ligne via Issuu : http://issuu.com/fullcirclemagazine. N'hésitez surtout pas à partager et à noter le FCM, pour aider à le faire connaître ainsi qu' Ubuntu Linux.

Magzster - Vous pouvez aussi lire le Full Circle online via Magzster : http://www.magzter.com/publishers/Full-Circle. N'hésitez surtout pas à partager et à noter le FCM, pour aider à le faire connaître ainsi qu'Ubuntu Linux.