Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume cinq

Parties 27 à 31

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 !

Pour l'instant, il s'agit d'une réédition simple de la série **Programmer en Python**, parties 27 à 31, des numéros 53 à 59 ; et, oui, l'incomparable professeur Python, Greg Walters, a pris quelques jours de congé au cours de cette partie de la série !

Gardez à l'esprit la date de publication ; les versions actuelles du matériel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 27

Greg Walters

Si vous avez déjà fait la queue pour acheter un billet de cinéma, vous avez été dans une file d'attente. Si vous avez eu à attendre dans les bouchons aux heures de pointe, vous avez été dans une file d'attente. Si vous avez déjà attendu dans un bureau administratif avec l'un de ces petits billets qui dit que vous êtes le numéro 98 et le panneau qui affiche « Numéro actuel : 42 », vous avez été dans une file d'attente.

Dans le monde des ordinateurs, les files d'attente sont très répandues. En tant qu'utilisateur, la plupart du temps vous n'avez pas à vous en préoccuper. Elles sont invisibles pour l'utilisateur. Mais si jamais vous avez à faire face à des événements en temps réel, vous allez finir par avoir à traiter avec elles. Il s'agit simplement de données d'un type ou d'un autre, qui attendent dans la file leur tour d'être traitées. Une fois qu'elles sont dans la file, elles attendent jusqu'à être traitées puis disparaissent. Vous ne pouvez pas connaître la valeur de l'élément de donnée suivant, sauf si vous le sortez de la file d'attente. Vous ne pouvez pas, par exemple, obtenir la valeur du quinzième élément de la file d'attente : il vous faut d'abord accéder aux 14 autres éléments. Une fois qu'un élément est consulté, il sort de la file d'attente. Il a disparu et il n'y a aucun moyen de récupérer les données à moins de les enregistrer dans une variable à long terme.

Il existe plusieurs types de files d'attente. Les plus courantes sont FIFO (« First In, First Out » ou premier entré, premier sorti), LIFO (« Last In, First Out » ou dernier entré, premier sorti), priorité et anneau. Nous parlerons des files d'attente anneau une autre fois.

Les files d'attente FIFO sont celles que nous voyons dans la vie quotidienne. Tous les exemples que j'ai énumérés ci-dessus sont des files d'attente FIFO. La première personne dans la ligne est traitée d'abord, s'en va, puis tout le monde se déplace d'une place dans la ligne. Dans un tampon FIFO, il n'y a pas de limite (sauf celle de la raison) au nombre d'éléments qu'il peut contenir. Ils s'empilent simplement dans l'ordre. Lorsqu'un élément est traité, il est sorti de la file et tous les autres se rapprochent d'une position du début de la file d'attente.

Les files d'attente LIFO sont moins fréquentes dans la vie, mais il existe encore des exemples réels. Celui qui vient tout de suite à l'esprit est l'exemple d'une pile d'assiettes dans votre placard de cuisine. Lorsque les assiettes sont lavées et séchées, elles s'empilent dans le placard. La dernière arrivée sur la pile est la première qui sera réutilisée. Tout le reste attend, peut-être pendant des jours, pour être utilisé. C'est une bonne chose que la file d'attente pour un billet de cinéma soit FIFO, n'est-ce pas ? Comme pour la file d'attente FIFO, en restant dans des tailles raisonnables, il n'y a pas de limite à la taille d'une file d'attente LIFO. Le premier élément entré dans la pile doit attendre que tous les éléments arrivés après lui soient retirés de la mémoire tampon (assiettes retirées de la pile) jusqu'à ce qu'il soit le seul restant.

Les files d'attente prioritaires sont un peu plus difficiles à comprendre du premier coup pour beaucoup de gens. Pensez à une entreprise qui possède une seule imprimante. Tout le monde utilise cette imprimante unique. Les travaux d'impression sont traités par ordre de priorité des départements. La paie a une priorité plus élevée (et heureusement) que, par exemple, vous, un programmeur. Vous avez une priorité plus élevée (et heureusement) que la réceptionniste. En bref, donc, les données qui ont une priorité plus élevée sont traitées et sortent de la file d'attente avant les données qui ont une priorité inférieure.

> Il existe plusieurs types de files d'attente. Les plus courantes sont FIFO (First In, First Out), LIFO (Last In, First Out), Priorité et Anneau.

FIFO

Les files d'attente FIFO sont faciles à visualiser en termes de données. Une liste Python est une représentation mentale facile. Considérez cette liste :

```
[1,2,3,4,5,6,7,8,9,10]
```

Il y a 10 articles dans la liste. En tant que liste, vous y accédez par l'index. Cependant, dans une file d'attente, vous ne pouvez pas accéder aux éléments par leur index. Vous devez traiter avec le prochain dans la file et la liste n'est pas figée. Elle est TRÈS dynamique. Lorsque nous demandons à accéder à l'élément suivant, il est retiré de la file d'attente.

```
import Queue
fifo = Queue.Queue()
for i in range(5):
    fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

Donc, en utilisant l'exemple ci-dessous, vous demandez un élément de la file d'attente. Elle retourne le premier élément (1) et la file d'attente ressemble alors à ceci :

```
[2,3,4,5,6,7,8,9,10]
```

Demandez-en deux de plus et vous obtenez 2, puis 3, et la file d'attente ressemble à ceci :

```
[4,5,6,7,8,9,10]
```

Je suis sûr que vous voyez l'idée. Python fournit une simple bibliothèque, assez étonnamment appelée « Queue » [Ndt : qui signifie file d'attente], qui fonctionne bien pour des files d'attente de petite et moyenne taille, jusqu'à environ 500 éléments. Voici un exemple simple de démonstration (encadré ci-dessus, première colonne).

Dans cet exemple, on initialise la file d'attente (fifo = Queue.Queue()) puis on y place les nombres de 0 à 4 (fifo.put(i)). Nous utilisons ensuite la méthode interne .get() pour retirer des éléments de la file

```
import Queue

fifo = Queue.Queue(12)
for i in range(13):
    if not fifo.full():
        fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

d'attente jusqu'à ce que la file d'attente soit vide, .empty(). Nous obtenons 0,1,2,3,4. Vous pouvez également définir le nombre maximal d'éléments que la file d'attente peut manipuler en l'initialisant avec la taille de la file d'attente comme cela :

```
fifo = Queue.Queue(300)
```

Une fois le nombre maximum d'éléments atteint, la file d'attente bloque toutes les entrées supplémentaires. Cela a cependant pour effet secondaire que le programme semble alors « planté ». La meilleure façon de contourner ce problème est d'utiliser la vérification Queue.full() qui indique si la file est pleine (encadré ci-dessus, deuxième colonne).

Ici, la file d'attente est paramétrée à un maximum de 12 éléments. Lorsque nous ajoutons des éléments dans la file d'attente, nous commençons avec 0 et arrivons à 11. Mais lorsque nous atteignons le nombre 12, le tampon est déjà plein. Puisque nous vérifions si la mémoire tampon est pleine avant d'essayer d'ajouter un élément, le dernier élément est tout simplement rejeté.

Il existe d'autres options, mais elles peuvent causer d'autres effets secondaires, et nous aborderons la question dans un prochain article. Ainsi, la plupart du temps, la voie à suivre est soit d'utiliser une file d'attente sans aucune limite, soit de s'assurer que l'on prévoit plus d'espace dans la file d'attente que ce dont on aura besoin.

LIFO

```
import Queue
lifo = Queue.LifoQueue()
for i in range(5):
    lifo.put(i)
while not lifo.empty():
    print lifo.get()
```

La bibliothèque « Queue » prend également en charge les files d'attente LIFO. Nous allons utiliser la liste ci-dessus comme exemple visuel. Lors de la mise en place de notre file d'attente, elle ressemble à ceci :

```
[1,2,3,4,5,6,7,8,9,10]
```

Si on retire trois éléments de la file d'attente, elle ressemble alors à ceci :

```
[1,2,3,4,5,6,7]
```

N'oubliez pas que dans une file d'attente LIFO, les éléments sont enlevés en commençant par le dernier entré. Voici l'exemple simple modifié pour une file d'attente LIFO (encadré troisième colonne).

Lorsqu'on l'exécute, on obtient 4,3,2,1,0.

Comme pour la file FIFO, vous pouvez régler la taille maximum de la file d'attente et utiliser la vérification .full() pour savoir si elle est pleine.

PRIORITÉ

Même si elle n'est pas souvent utilisée, une file de priorité peut parfois être utile. C'est à peu près la même structure que pour les autres files d'attente, mais nous devons lui passer un tuple qui contient à la fois la priorité et les données. Voici un exemple en utilisant la bibliothèque « Queue » :

```
pq = Queue.PriorityQueue()
pq.put((3,'Moyenne 1'))
pq.put((4,'Moyenne 2'))
pq.put((10,'Basse'))
pq.put((1,'Haute'))

while not pq.empty():
    suiv = pq.get()
    print suiv
    print suiv[1]
```

D'abord on initialise la file d'attente.

Puis nous y plaçons quatre éléments. Remarquez que nous utilisons le format (priorité, données) pour placer nos données. La bibliothèque trie nos données selon un ordre basé sur la valeur de priorité. Quand nous extrayons les données, elles ressortent sous forme de tuple, comme lors de l'insertion. Vous pouvez utiliser l'indice pour accéder aux deux parties du tuple. Voici ce que nous obtenons :

```
(1, 'Haute')
Haute
(3, 'Moyenne 1')
Moyenne 1
(4, 'Moyenne 2')
Moyenne 2
(10, 'Basse')
Basse(
```

Dans nos deux premiers exemples, nous avons simplement affiché les données qui sortent de notre file d'attente. C'est très bien pour ces exemples, mais dans le monde réel de la programmation, vous aurez probablement besoin de faire quelque chose avec cette information dès qu'elle sort de la file d'attente, sinon elle sera perdue. Lorsque nous utilisons « print fifo.get », nous envoyons les données vers le terminal puis elles sont détruites. Il faut juste garder ça à l'esprit.

Maintenant, nous allons utiliser une partie de ce que nous avons déjà appris sur Tkinter pour créer un programme de démo de file d'attente. Cette démo aura deux cadres. Le premier contiendra (pour l'utilisateur) trois boutons. Un pour une file d'attente FIFO, un pour une file d'attente LIFO, et un autre pour une file de priorité. Le second cadre contiendra un widget champ de texte, deux boutons, l'un pour ajouter à la file d'attente et l'autre pour retirer de la file, et trois labels, l'un montrant quand la file est vide, l'un montrant quand la file est pleine, et un dernier pour afficher ce qui a été retiré de la file d'attente. Nous allons également écrire du code pour centrer automatiquement la fenêtre sur l'écran. Voici le début du code (encadré ci-dessus haut de la deuxième colonne).

Ici, nous avons nos importations et le début de notre classe. Comme précédemment, nous créons la routine init avec les routines DefinirVariables, Cons-

```
import sys
from Tkinter import *
import ttk
import tkMessageBox
import Queue

class TestFiles:
  def __init__(self,principale = None):
      self.DefinirVariables()
      f = self.ConstruireWidgets(principale)
      self.PlacerWidgets(f)
      self.AfficherStatut()
```

```
def DefinirVariables(self):
      self.TypeDeFile = ''
      self.StatutPlein = StringVar()
      self.StatutVide = StringVar()
      self.Element = StringVar()
      self.Sortie = StringVar()
      # Definit les files
      self.fifo = Queue.Queue(10)
      self.lifo = Queue.LifoQueue(10)
      self.pq = Queue.PriorityQueue(10)
      self.obj = self.fifo
```

```
def ConstruireWidgets(self,principale):
  # Definit nos widgets
  fenetre = Frame(principale)
  self.f1 = Frame(fenetre,
      relief = SUNKEN,
      borderwidth=2,
      width = 300,
      padx = 3,
      pady = 3
  )
  self.btnFifo = Button(self.f1,
      text = "FIFO"
  )

  self.btnFifo.bind('<Button-1>',
      lambda e: self.btnMain(1)
  )
  self.btnLifo = Button(self.f1,
      text = "LIFO"
  )
  self.btnLifo.bind('<ButtonRelease-1>',
      lambda e: self.btnMain(2)
  )
  self.btnPriority = Button(self.f1,
      text = "PRIORITY"
  )
  self.btnPriority.bind('<ButtonRelease-1>',
      lambda e: self.btnMain(3)
  )
```

truireWidgets et PlacerWidgets. Nous avons aussi une routine appelée AfficherStatut qui… affichera l'état de notre file d'attente (encadré page précédente, en haut au milieu).

Nous allons maintenant créer notre routine DefinirVariables. Nous avons quatre objets StringVar(), une variable vide appelée TypeDeFile, et trois objets file d'attente - un pour chaque type de file d'attente avec lesquels nous allons jouer. Nous avons fixé la taille maximale des files d'attente à 10 pour les besoins de la démo. Nous avons aussi créé un objet appelé obj auquel nous assignons la valeur FIFO. Lorsque nous sélectionnerons un type de file avec les boutons, nous mettrons dans cet objet le type de file d'attente que nous voulons. De cette façon, une file d'attente est conservée quand on passe à un autre type de file d'attente (encadré page précédente, en haut à droite).

Ici nous commençons la définition des widgets. Nous créons notre premier cadre, les trois boutons et leurs fonctions de rappel. Notez que nous utilisons la même routine pour gérer les fonctions de rappel. Chaque bouton envoie une valeur à la routine de rappel pour indiquer quel bouton a été cliqué. Nous aurions tout aussi bien pu créer une routine dédiée pour chaque bouton. Cependant, puisque les trois boutons gèrent une tâche commune, j'ai pensé qu'il serait bon de les considérer comme un groupe (page précédente, en bas à droite).

Ensuite nous mettons en place le second cadre, le widget de saisie et les deux boutons. La seule chose ici qui sort de l'ordinaire est le rappel pour le widget de saisie. Ici nous associons la routine self.AjouterALaFile à la touche « Return » (Entrée). De cette façon, l'utilisateur n'a pas à utiliser la souris pour ajouter les données. Il peut simplement entrer les données dans la zone de saisie et appuyer sur Entrée (encadré ci-contre, en haut).

Voici les trois dernières définitions de widgets. Toutes les trois sont des étiquettes. Nous réglons l'attribut textvariable des variables que nous avons définies plus tôt. Si vous vous souvenez, lorsque cette variable change, le texte de l'étiquette changera aussi. Nous faisons aussi quelque chose d'un peu différent sur l'étiquette lblData. Nous allons utiliser une police différente pour faire ressortir l'affichage des données extraites de la file d'attente. Rappelez-vous que nous devons retourner l'objet fenêtre de sorte qu'il puisse être utilisé dans la routine PlacerWidgets (ci-contre en bas).

C'est le début de la routine PlacerWidgets. Remarquez que nous avons mis ici cinq étiquettes vides tout en haut de la fenêtre racine. Je fais cela pour régler

```
self.f2 = Frame(fenetre,
    relief = SUNKEN,
    borderwidth=2,
    width = 300,
    padx = 3,
    pady = 3
)
self.txtAdd = Entry(self.f2,
    width=5,
    textvar=self.Element
)
self.txtAdd.bind('<Return>',self.AjouterALaFile)
self.btnAdd = Button(self.f2,
    text='Ajout dans la file',
    padx = 3,
    pady = 3
)
self.btnAdd.bind('<ButtonRelease-1>',self.AjouterALaFile)
self.btnGet = Button(self.f2,
    text='Recupere element suivant',
    padx = 3,
    pady = 3
)
self.btnGet.bind('<ButtonRelease-1>',self.RecupererDansFile)
```

```
    self.lblEmpty = Label(self.f2,
        textvariable=self.StatutVide,
        relief=FLAT
    )
    self.lblFull = Label(self.f2,
        textvariable=self.StatutPlein,
        relief=FLAT
    )
    self.lblData = Label(self.f2,
        textvariable=self.Sortie,
        relief = FLAT,
        font=("Helvetica", 16),
        padx = 5
    )

    return fenetre
```

l'espacement. C'est un moyen facile de « tricher » pour faciliter le placement de la fenêtre. Nous réglons ensuite le premier cadre, puis une autre étiquette « de triche », puis les trois boutons .

Nous plaçons maintenant le deuxième cadre, encore une étiquette « de triche » puis le reste de nos widgets.

Ensuite nous avons notre routine « standard » pour quitter l'application, qui appelle simplement sys.exit() :

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

Maintenant, notre routine principale de rappel pour les boutons, btnMain. Rappelez-vous que nous lui envoyons (via le paramètre p1) quel bouton a été cliqué. Nous utilisons la variable self.TypeDeFile en référence au type de file d'attente que nous sommes en train de gérer, puis nous assignons à self.obj la file d'attente appropriée et, enfin, changeons le titre de notre fenêtre racine pour afficher le type de file d'attente que nous utilisons. Après cela, nous affichons le type de file dans le terminal (vous n'êtes pas obligé de faire cela), puis appelons la routine AfficherStatut. Maintenant nous allons écrire la routine AfficherStatut (page suivante, encadré en haut, à droite).

Comme vous pouvez le voir, c'est assez simple. Nous réglons les variables d'étiquettes à leur bon état afin qu'elles

```
def btnMain(self,p1):
    if p1 == 1:
        self.TypeDeFile = 'FIFO'
        self.obj = self.fifo
        root.title('Tests Files - FIFO')
    elif p1 == 2:
        self.TypeDeFile = 'LIFO'
        self.obj = self.lifo
        root.title('Tests Files - LIFO')
    elif p1 == 3:
        self.TypeDeFile = 'PRIORITY'
        self.obj = self.pq
        root.title('Tests Files - Priorite')
    print self.TypeDeFile
    self.AfficherStatut()
```

```
    self.f2.grid(column = 0,row = 2,sticky='nsew',columnspan=5,padx = 5, pady = 5)
    l = Label(self.f2,text='',width = 15,anchor = 'e').grid(column = 0, row = 0)
    self.txtAdd.grid(column=1,row=0)
    self.btnAdd.grid(column=2,row=0)
    self.btnGet.grid(column=3,row=0)
    self.lblEmpty.grid(column=2,row=1)
    self.lblFull.grid(column=3,row = 1)
    self.lblData.grid(column = 4,row = 0)
```

```
def PlacerWidgets(self, principale):
    fenetre = principale
    # Place les widgets
    fenetre.grid(column = 0, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 0, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 1, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 2, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 3, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 4, row = 0)

    self.f1.grid(column = 0,row = 1,sticky='nsew',columnspan=5,padx = 5,pady = 5)
    l = Label(self.f1,text='',width = 25,anchor = 'e').grid(column = 0, row = 0)
    self.btnFifo.grid(column = 1,row = 0,padx = 4)
    self.btnLifo.grid(column = 2,row = 0,padx = 4)
    self.btnPriority.grid(column = 3, row = 0, padx = 4)
```

affichent si la file d'attente que nous utilisons est pleine, vide, ou quelque part entre les deux.

La routine AjouterALaFile est également assez simple. Nous récupérons les données du champ de saisie en utilisant la fonction .get(). Nous vérifions ensuite si le type courant de file d'attente est une file de priorité. Si c'est le cas, nous devons nous assurer que le format de saisie est correct. Nous vérifions cela en testant la présence d'une virgule. S'il n'y en a pas, nous prévenons l'utilisateur via une boîte de message d'erreur. Si tout semble correct, nous vérifions ensuite si la file d'attente que nous utilisons actuellement est pleine (encadré ci-contre, en bas). N'oubliez pas, si la file est pleine, la routine d'insertion est bloquée et le programme va planter. Si tout va bien, nous ajoutons l'élément à la file d'attente et mettons à jour le statut.

La routine RecupererDansFile est encore plus facile. Nous vérifions si la file est vide afin de ne pas nous heurter à un problème de blocage et, si ce n'est pas le cas, nous retirons les données de la file d'attente, l'affichons, et mettons à jour le statut (encadré ci-contre, au milieu).

Nous arrivons à la fin de notre application. Voici la routine de centrage de fenêtre. Nous récupérons d'abord la largeur et la hauteur de l'écran. Nous récupérons ensuite la largeur et la hauteur de la fenêtre racine à l'aide des routines winfo_reqwidth() et winfo_reqheight() intégrées à tkinter. Ces routines, lorsqu'elles sont appelées au bon moment, retourneront la largeur et la hauteur de la fenêtre racine en tenant compte du placement des widgets. Si vous l'appelez trop tôt, vous obtiendrez des valeurs, mais pas celles dont vous avez vraiment besoin. Nous soustrayons ensuite la largeur de la fenêtre de la largeur de l'écran, et divisons cela par 2, puis nous faisons la même chose pour la hauteur. Nous utilisons alors ces informations

```
if name == 'main':
  def Centrer(window):
      # recupere largeur et hauteur de l'ecran
      largeurE = window.winfo_screenwidth()
      hauteurE = window.winfo_screenheight()
      # recupere largeur et hauteur de la fenetre
      largeurF = window.winfo_reqwidth()
      hauteurF = window.winfo_reqheight()
      xc = (largeurE-largeurF)/2
      yc = (hauteurE-hauteurF)/2

window.geometry("%dx%d+%d+%d"%(largeurF,hauteurF,xc,yc))
      window.deiconify()
```

```
def AfficherStatut(self):
    # verifie si vide
    if self.obj.empty() == True:
        self.StatutVide.set('Vide')
    else:
        self.StatutVide.set('')
    # verifie si plein
    if self.obj.full() == True:
        self.StatutPlein.set('Plein')
    else:
        self.StatutPlein.set('')
```

```
def RecupererDansFile(self,p1):
    self.Sortie.set('')
    if not self.obj.empty():
        temp = self.obj.get()
        self.Sortie.set("Sorti
{0}".format(temp))
    self.AfficherStatut()
```

```
def AjouterALaFile(self,p1):
    temp = self.Element.get()
    if self.TypeDeFile == 'PRIORITY':
        commapos = temp.find(',')
        if commapos == -1:
            print "ERREUR"
            tkMessageBox.showerror('Demo File',
                'Un element Priority doit etre au
format\r(priorite,valeur)')
        else:
            self.obj.put(self.Element.get())
    elif not self.obj.full():
        self.obj.put(self.Element.get())
    self.Element.set('')
    self.AfficherStatut()
```

dans l'appel de la fonction geometry. La plupart du temps, cela fonctionne à merveille. Toutefois, il pourrait y avoir des moments où vous aurez besoin de définir la largeur et la hauteur à la main (encadré haut de la deuxième colonne, page précédente).

```
root = Tk()
root.title('Tests File - FIFO')
demo = TestFiles(root)
root.after(3,Centrer,root)
root.mainloop()
```

Enfin, nous instancions la fenêtre racine, définissons le titre de base et instancions la class TestFiles. Nous appelons ensuite root.after, qui attend un nombre x de millisecondes (dans ce cas 3), après que la fenêtre racine soit instanciée, puis appelle la routine Centrer. De cette façon, la fenêtre racine a été complètement paramétrée et est prête à s'afficher, donc nous pouvons obtenir sa largeur et sa hauteur. Vous pourriez avoir à ajuster légèrement le temps de retard. Certaines machines sont beaucoup plus rapides que d'autres. 3 fonctionne très bien sur ma machine, votre réglage peut varier. Enfin nous appelons la boucle principale de la fenêtre racine pour exécuter l'application.

Pendant que vous jouez avec les files d'attente, notez que si vous mettez des données dans une file d'attente (disons la file d'attente FIFO), puis passez à une autre file d'attente (disons la file d'attente LIFO), les données qui ont été placées dans la file FIFO sont toujours là et vous attendent. Vous pouvez complètement ou partiellement remplir les trois files d'attente, puis commencer à jouer avec.

Eh bien, c'est tout pour cette fois-ci. Amusez-vous avec vos files d'attente. Le code de TestFiles peut être trouvé ici : http://pastebin.com/MKLTmSES.

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.com.

TUTORIEL

Programmer en Python - Partie 28

Greg Walters

Nous allons approfondir l'exploration des widgets fournis par Tkinter. Cette fois, nous allons examiner les menus, listes déroulantes, les boîtes de sélection, barres de séparation, barres de progression et les onglets. Nous en parlerons à tour de rôle.

Vous avez vu des menus dans presque toutes les applications que vous utilisez. Tkinter rend très facile la création des menus. Les listes déroulantes sont similaires aux listes dont nous avons parlé dans le dernier article sur la démo des widgets, sauf que la liste se déroule vers le bas au lieu d'être visible en permanence. Les contrôles de sélection numérique sont pratiques pour définir une plage fixe de valeurs dans laquelle on peut se déplacer vers le haut ou vers le bas. Par exemple, si nous voulons que l'utilisateur soit en mesure de choisir des entiers compris entre 1 et 100, nous pouvons facilement utiliser une boîte de sélection. Les barres de progression sont une merveilleuse façon de montrer que votre application n'a pas planté quand quelque chose prend beaucoup de temps, comme la lecture des enregistrements d'une base de données. Elles peuvent montrer le pourcentage d'achèvement d'une tâche. Il y a deux types de barres de progression, déterminée et indéterminée. Vous utilisez une barre de progression déterminée quand vous savez exactement combien d'actions vous devez réaliser. Si vous ne connaissez pas le nombre d'actions ou le pourcentage de progression de votre tâche à un instant t, vous pouvez utiliser la version indéterminée. Nous allons travailler avec les deux. Enfin, un widget à onglets verticaux (ou widget à onglets horizontaux) est régulièrement utilisé pour les réglages des écrans de configuration. Vous pouvez regrouper logiquement une série de widgets sur chaque onglet.

Nous allons donc commencer. Comme d'habitude, nous allons créer une application de base et construire notre programme avec des widgets supplémentaires, que nous allons lui ajouter. Regardez à droite pour la première partie de notre application. Vous avez déjà vu presque tout cela.

Enregistrez le code en tant que widgetdemo2a.py. Rappelez-vous, nous allons l'utiliser comme base pour construire la démonstration complète. Maintenant, nous allons commencer le processus de création du menu. Voici les étapes que nous allons suivre. Premièrement, nous définissons une variable pour contenir l'occurrence de menu. Comme la plupart des widgets que nous utilisons, le format est :

```
NotreVariable =
Widget(parent, options).
```

```
import sys
from Tkinter import *
import ttk
# Montre comment créer un menu
class WidgetDemo2:

    def __init__(self,principale = None):
        self.DefinirVariables()
        f = self.ConstruireWidgets(principale)
        self.PlacerWidgets(f)

    def DefineVars(self):
        pass
```

Et voici la fin de notre programme. Vous avez déjà vu ça précédemment, rien de nouveau ici.

```
if __name__ == '__main__':
    def Center(window):
      # recupere largeur et hauteur de l'ecran
      sw = window.winfo_screenwidth()
      sh = window.winfo_screenheight()
      # recupere largeur et hauteur de la fenetre
      rw = window.winfo_reqwidth()
      rh = window.winfo_reqheight()
      xc = (sw-rw)/2
      yc = (sh-rh)/2
      print "{0}x{1}".format(rw,rh)
        window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
        window.deiconify()

  root = Tk()
  root.title('Demo de plus de widgets')
  demo = DemoWidget2(root)
  root.after(13,Center,root)
  root.mainloop()
```

Dans le cas présent, nous utilisons le widget Menu avec l'attribut « principale » en tant que fenêtre-mère. Nous faisons cela dans la routine ConstruireWidgets. Ensuite, nous créons un autre élément de menu, cette fois-ci en le nommant menuFichier. Nous ajouterons des commandes et des séparateurs, au besoin. Enfin, nous l'ajoutons à la barre de menu et continuons de la sorte jusqu'à ce que nous ayons fini. Dans notre exemple, nous allons avoir la barre de menu, un menu déroulant Fichier, un menu déroulant Edition et un menu déroulant Aide (en haut à droite). Commençons.

Ensuite (au milieu à droite), nous nous concentrons sur le menu Fichier. Il contiendra cinq éléments. Nouveau, Ouvrir, Sauver, un séparateur et Quitter. Nous allons utiliser la méthode .add_command pour ajouter les commandes. Tout ce que nous devons faire, c'est appeler la méthode avec le texte (label =) et ensuite fournir une fonction de rappel pour prendre la main quand l'utilisateur clique sur l'élément. Enfin, nous utilisons la fonction menubar.add_cascade pour attacher le menu à la barre.

Notez que la commande Quitter utilise « root.quit » pour mettre fin au programme. Pas besoin de fonction de rappel pour cela. Ensuite, nous ferons la même chose pour les menus Edition et Aide.

Notez la partie « tearoff = 0 » dans chacune des définitions de groupe de menu. Si vous changez le 0 en 1, le menu commencera par une sorte de ligne pointillée qui permet de détacher le menu de la barre de menus en créant sa propre fenêtre. Bien que cela puisse être utile dans le futur, ce n'est pas ce que nous voulons ici.

Enfin et surtout, nous devons placer le menu. Nous ne faisons pas un placement normal avec la fonction .grid(). Nous allons simplement l'ajouter en utilisant la fonction parent.config (en bas à droite).

Tout cela est placé dans la routine ConstruireWidgets. Maintenant (page suivante, en haut à droite), nous avons besoin d'ajouter un cadre générique et de mettre l'instruction de retour avant de passer à la routine PlacerWidgets.

Enfin (page suivante, en bas), nous devons créer toutes les fonctions de rappel que nous avons définies plus tôt. Pour la démo, nous allons simplement afficher quelque chose dans le terminal utilisé pour lancer le programme.

C'est tout. Enregistrez et exécutez le programme. Cliquez sur chacune des options de menu (en gardant Fichier|Quitter pour la fin).

```
def ConstruireWidgets(self,principale):
  fenetre = Frame(principale)
    #==============================
    #       LES MENUS
    #==============================
    # Creation de la barre de menus
  self.barreMenus = Menu(principale)
```

```
# Creation du menu Fichier et ajout a la barre de menus
menuFichier = Menu(self.barreMenus, tearoff = 0)
menuFichier.add_command(label = "Nouveau", command = self.FichierNouveau)
menuFichier.add_command(label = "Ouvrir", command = self.FichierOuvrir)
menuFichier.add_command(label = "Sauver", command = self.FichierSauver)
menuFichier.add_separator()
menuFichier.add_command(label = "Quitter", command = root.quit)
self.barreMenus.add_cascade(label = "Fichier", menu = menuFichier)
```

```
 # Creation du menu Edition
menuEdition = Menu(self.barreMenus, tearoff = 0)
menuEdition.add_command(label = "Couper", command = self.EditionCouper)
menuEdition.add_command(label = "Copier", command = self.EditionCopier)
menuEdition.add_command(label = "Coller", command = self.EditionColler)
self.barreMenus.add_cascade(label = "Edition", menu = menuEdition)
# Creation du menu Aide
menuAide = Menu(self.barreMenus, tearoff=0)
menuAide.add_command(label = "A propos", command = self.AideApropos)
self.barreMenus.add_cascade(label = "Aide", menu = menuAide)
```

```
  # affichage du menu
 principale.config(menu = self.barreMenus)
  #=======================================
  #           FIN DES MENUS
  #=======================================
```

Maintenant (ci-dessous), nous allons traiter la liste déroulante. Enregistrez votre fichier sous widgetdemo2b.py et nous serons prêts à commencer. Les importations, les définitions de classes et la routine __init__ sont toutes les mêmes, ainsi que la partie inférieure du programme. Nous allons ajouter deux lignes à la routine DefinirVariables. Commentez ou effacez l'instruction « pass » et mettez le code suivant (j'ai inclus la ligne de définition juste pour la clarté).

Nous définissons d'abord une étiquette, comme nous l'avons déjà fait. Ensuite, nous définissons la liste déroulante. Nous utilisons « ttk.Combobox », définissons le parent et réglons la hauteur à 19, la largeur à 20 et le textvariable à « self.selectionListeDeroulante1 ». Rappelez-vous que nous avons utilisé les « textvariables » dans le dernier article, mais juste au cas où vous l'auriez oublié… il change à tout moment sa valeur dès que la liste déroulante est modifiée. Nous l'avons défini dans DefinirVariables comme un objet StringVar. Ensuite nous chargeons les valeurs que nous voulons que l'utilisateur puisse choisir, et de nouveau, nous les avons définies dans DefinirVariables. Enfin, nous lions l'événement virtuel « ComboboxSelected » à la routine testListeDeroulante que nous allons étoffer dans une minute.

Ensuite, nous allons placer la liste déroulante et le titre dans notre fenêtre (page suivante en haut à droite).

```
        self.f1 = Frame(fenetre,
                        relief = SUNKEN,
                        borderwidth = 2,
                        width = 500,
                        height = 100
                        )

        return fenetre
```

Ensuite (comme nous l'avons déjà fait) nous nous occupons de placer les autres widgets.

```
    def PlacerWidgets(self,principale):
      fenetre = principale
      fenetre.grid(column = 0, row = 0)

      self.f1.grid(column = 0,
                   row = 0,
                   sticky = 'nsew'
                   )
```

```
    def DefinirVariables(self):
      self.selectionListeDeroulante1 = StringVar()
      self.valeursC1 = ['Neant','Option 1','Option 2','Option 3']
```

Insérez le code suivant dans ConstruireWidgets après la définition self.f1 et avant la ligne « return fenetre ».

```
      self.labelListeDeroulante = Label(self.f1, text = "Liste deroulante : ")
      self.listeDeroulante1 = ttk.Combobox(self.f1,
                              height = "19",
                              width = "20",
                              textvariable = self.selectionListeDeroulante1
                             )
      self.listeDeroulante1['values'] = self.valeursC1
      # associe l'evenement virtuel a une fonction de rappel

self.listeDeroulante1.bind("<<ComboboxSelected>>",self.testListeDeroulante)
```

```
def FichierNouveau(self):
    print "Menu - Fichier Nouveau"
def FichierOuvrir(self):
    print "Menu - Fichier Ouvrir"
def FichierSauver(self):
    print "Menu - Fichier Sauver"
def EditionCouper(self):
    print "Menu - Edition Couper"
def EditionCopier(self):
    print "Menu - Edition Copier"
def EditionColler(self):
    print "Menu - Edition Coller"
def AideApropos(self):
    print "Menu - Aide Apropos"
```

Sauvegardez tout et testez.

Maintenant enregistrez sous widgetdemo2c.py et nous allons commencer avec la barre de séparation. C'est super facile. Alors que les mises à jour de Tkinter fournissent un widget barre de séparation, je n'ai jamais été en mesure de le faire fonctionner. Voici une autre façon de faire. Nous utiliserons un cadre avec une hauteur de 2. Les seuls changements à notre programme seront la définition du cadre dans ConstruireWidgets après l'instruction « bind » de la liste déroulante et le placement du cadre dans la routine PlacerWidgets. Donc dans ConstruireWidgets ajoutez les lignes suivantes (montrées au milieu à droite).

Vous avez déjà vu tout cela avant. Enregistrez et testez. Vous aurez probablement à élargir la fenêtre de premier niveau pour voir le séparateur, tout cela va être bien plus évident dans la prochaine démo. Enregistrez en tant que widgetdemo2d.py, nous allons ajouter la zone de sélection numérique.

Sous DefinirVariables, ajoutez la ligne suivante :

```
self.valeurSelection =
StringVar()
```

Maintenant, vous savez que c'est pour pouvoir obtenir sa valeur à tout moment. Ensuite, nous allons ajouter du code à la routine ConstruireWidgets, juste avant la ligne « return fenetre » (en bas à droite).

Ici, nous définissons une étiquette et la zone de sélection numérique. La définition de la zone de sélection est :

```
notreWidget =
Spinbox(parent,valeur basse,
valeur haute, largeur,
textvariable, wrap)
```

La valeur mini doit s'appeler « from_ » car le mot « from » est un mot-clé réservé du langage python et l'utiliser risquerait de casser votre programme. Les valeurs « from_ » et « to » doivent être définies comme valeurs flottantes. Ici, nous voulons que la valeur mini soit 1 et la valeur maxi soit 10. Enfin l'option « wrap » signifie que si la valeur est (dans notre cas) 10 et que l'utilisateur clique sur la flèche du haut, nous voulons qu'il retourne à la valeur mini et ainsi de suite. Il en est de même pour la valeur mini. Si l'utilisateur clique sur la flèche du bas alors que la valeur est 1, il retourne à 10 et ainsi de suite. Si vous mettez « wrap = False », le mécanisme s'arrête simplement et il n'y a pas de bouclage.

Maintenant, nous allons placer les widgets dans PlacerWidgets (page suivante en bas à gauche).

Voilà, c'est tout. Enregistrez et jouez. Vous voyez nettement le séparateur maintenant. Enregistrez en tant que widgetdemo2e.py et nous allons créer les barres de progression.

```
self.labelSelection.grid(column = 0, row = 4)
self.selection1.grid(column = 1,
                     row = 4,
                     pady = 2
                       )
```

Et enfin on écrit la fonction de retour qui affiche simplement dans le terminal ce que l'utilisateur a choisi.

```
def testListeDeroulante(self,p1):
print self.selectionListeDeroulante1.get()
```

```
self.separ = Frame(self.f1,
                   width = 140,
                   height = 2,
                   relief = RIDGE,
                   borderwidth = 2
                   )
```

Puis ajoutez cela dans PlacerWidgets…

```
self.separ.grid(column = 0,
                row = 3,
                columnspan = 8,
                sticky = 'we',
                padx = 3,
                pady = 3
                )
```

```
self.labelSelection = Label(self.f1, text = "Selection numerique :")
self.selection1 = Spinbox(self.f1,
                          from_ = 1.0,
                          to = 10.0,
                          width = 3,
                          textvariable = self.valeurSelection,
                          wrap=True
                          )
```

Encore une fois, nous avons besoin de définir certaines variables, dans la routine DefinirVariables ajoutez le code suivant :

```
 self.valeurSelection2 =
StringVar ()
self.boutonEtat = False
self.valeurBarreProg2 =
StringVar ()
```

Il est assez évident de deviner ce que sont les deux variables StringVar. Nous parlerons de self.boutonEtat dans un instant. Pour le moment, continuons et définissons les widgets pour cette portion dans Construire-Widgets (à droite).

De nouveau ceci est placé avant le « return fenetre ». Ce que nous faisons c'est la mise en place d'un cadre qui contiendra les widgets. Puis, nous avons mis en place deux étiquettes comme guides. Et nous définissons la première barre de progression. Les seules choses qui pourraient être étranges sont la longueur, le mode et le maximum. La longueur est la taille en pixels de notre barre. Le maximum est la valeur la plus élevée possible. Dans ce cas, c'est 100 comme nous utilisons des pourcentages. Dans le cas présent mode vaut « indéterminé ». Rappelez-vous, nous utilisons ce mode lorsque nous ne savons pas précisément où nous en sommes dans la progression d'une tâche, mais que nous voulons que l'utilisateur sache qu'il se passe toujours quelque chose.

Maintenant, nous ajoutons un bouton (vous l'avez déjà fait), une autre étiquette, une autre barre de progression et une autre zone de sélection numérique. Le mode de cette seconde barre de progression est « déterminé ». Nous utilisons la zone de sélection numérique pour régler le « pourcentage » d'achèvement. Puis, ajoutons les lignes suivantes (page suivante, en haut à droite) dans la routine PlacerWidgets.

Finalement, nous ajoutons deux routines pour contrôler nos barres de progression (page suivante en bas à droite).

La routine TestBarreProg contrôle la barre de progression indéterminée. Simplement, nous démarrons et arrêtons une horloge interne qui est intégrée dans la barre de progression. La ligne « self.barreProg.start(10) » paramètre le minuteur à 10 millisecondes. Cela rend le mouvement de la barre assez rapide. N'hésitez pas à jouer avec cette valeur à la hausse ou à la baisse. La routine SelectionAction définit simplement l'avancement

```
self.labelSelection.grid(column = 0, row = 4)
self.selection1.grid(column = 1,
                     row = 4,
                     pady = 2
                     )
```

```
  #=======================================
  #        BARRE DE PROGRESSION
  #=======================================
self.lbl0 = Label(self.fBarreProg,
                  text = "Barres de progression"
                  )
self.lbl1 = Label(self.fBarreProg,
                  text = "Indeterminee",
                  anchor = 'e'
                  )
self.barreProg = ttk.Progressbar(self.fBarreProg,
                  orient = HORIZONTAL,
                  length = 100,
                  mode = 'indeterminate',
                  maximum = 100
                  )
self.btnptest = Button(self.fBarreProg,
                  text = "Demarrer",
                  command = self.TestBarreProg
                  )
self.lbl2 = Label(self.fBarreProg,
                  text = "Determinee"
                  )
self.barreProg2 = ttk.Progressbar(self.fBarreProg,
                  orient = HORIZONTAL,
                  length = 100,
                  mode = 'determinate',
                  variable = self.valeurBarreProg2
                  )
self.selection2 = Spinbox(self.fBarreProg,
                  from_ = 1.0,
                  to = 100.0,
                  textvariable = self.valeurSelection2,
                  wrap = True,
                  width = 5,
                  command = self.SelectionAction
                  )
```

de la barre de progression en fonction de la valeur sélectionnée. Nous l'affichons dans un terminal.

C'est tout pour le moment. Sauvegardez et jouez.

Maintenant sauvegardez sous le nom widgetdemo2f.py et nous allons nous occuper des onglets. Ajoutez le code suivant dans ConstruireWidgets (ci-dessous) avant la ligne « return fenetre » :

Regardons ce que nous avons fait. Premièrement, nous définissons un cadre pour notre widget « onglets ». Puis nous définissons le widget. Nous avons déjà rencontré toutes les options auparavant. Ensuite, nous définissons deux cadres nommés self.p1 et self.p2 qui seront nos pages. Les deux lignes suivantes (self.on-glets.add) attachent les cadres au widget et ils ont un onglet qui leur est rattaché. Nous avons également réglé les titres des onglets. Enfin, nous mettons une étiquette sur la page numéro un. Nous allons en mettre une sur la page deux lorsque nous plaçerons les contrôles juste pour le plaisir.

Dans la routine PlacerWidgets, insérez le code suivant (page suivante en bas à gauche).

```
        #=======================================
        #               ONGLETS
        #=======================================
        self.fenetreOnglets = Frame(self.f1,
                            relief = SUNKEN,
                            borderwidth = 2,
                            width = 500,
                            height = 300
                            )
      self.onglets = ttk.Notebook(self.fenetreOnglets,
                                  width = 490,
                                  height = 290
                                  )
      self.p1 = Frame(self.onglets)
      self.p2 = Frame(self.onglets)
      self.onglets.add(self.p1,text = 'Page 1')
      self.onglets.add(self.p2,text = 'Page 2')
      self.labelPage1 = Label(self.p1,
                          text = "Voici un texte sur la
page 1",
                          padx = 3,
                          pady = 3
                            )
```

```
  # Barre de progression
self.fBarreProg.grid(column = 0,
                     row = 5,
                     columnspan = 8,
                     sticky = 'nsew',
                     padx = 3,
                     pady = 3
                     )
self.lbl0.grid(column = 0, row = 0)
self.lbl1.grid(column = 0,
                    row = 1,
                    pady = 3
                    )
self.barreProg.grid(column = 1, row = 1)
self.btnptest.grid(column = 3, row = 1)
self.lbl2.grid(column = 0,
                    row = 2,
                    pady = 3
                    )
self.barreProg2.grid(column = 1, row = 2)
self.selection2.grid(column = 3, row = 2)
```

```
def TestBarreProg(self):
    if self.boutonEtat == False:
        self.btnptest.config(text="Arreter")
        self.boutonEtat = True
        self.barreProg.start(10)
    else:
        self.btnptest.config(text="Demarrer")
        self.boutonEtat = False
        self.barreProg.stop()

def SelectionAction(self):
    v = self.valeurSelection2.get()
    print v
    self.valeurBarreProg2.set(v)
```

La seule chose qui pourrait paraître étrange, c'est l'étiquette sur la page deux. Nous combinons la définition et la mise en place dans la grille avec la même commande. Nous l'avons déjà fait dans notre première application de démo.

C'est fini. Sauvegardez et amusez-vous.

Comme toujours le code de l'application complète est sur pastebin :
http://pastebin.com/7BJr54au.

Au plaisir. La prochaine fois nous allons plutôt aborder des trucs sur les bases de données.

```
    self.fenetreOnglets.grid(column = 0,
                             row = 6,
                             columnspan = 8,
                             rowspan = 7,
                             sticky = 'nsew'
                             )
    self.onglets.grid(column = 0,
                      row = 0,
                      columnspan = 11,
                      sticky = 'nsew'
                      )
    self.labelPage1.grid(column = 0,row = 0)
    self.labelPage2 = Label(self.p2,
                      text = 'Voici un texte sur la page 2',
                      padx = 3,
                      pady = 3
                      ).grid(
                             column = 0,
                             row = 1
                               )
```

TUTORIEL

Programmer en Python - Partie 29

Greg Walters

Il y a quelque temps, on m'a demandé de convertir une base de données MySQL en SQLite. En cherchant une solution rapide et facile (et gratuite) sur internet, je n'ai rien trouvé qui fonctionnait pour moi avec la version actuelle de MySQL. Alors j'ai décidé d'aller de l'avant et de fabriquer ma solution moi-même.

Le programme MySQL Administrator vous permet de sauvegarder une base de données dans un fichier texte à plat. Beaucoup de navigateurs SQLite vous permettent de lire un fichier SQL de définition à plat et de créer la base de données à partir de là. Cependant, il y a beaucoup de choses que MySQL supporte, mais pas SQLite. Alors ce mois-ci, nous allons écrire un programme de conversion qui lit un fichier de « dump » (sauvegarde) MySQL et crée une version SQLite.

Commençons par regarder le fichier de « dump » MySQL. Il se compose d'une section qui crée la base de données, puis des sections qui créent chaque table dans la base et insèrent des données dans ces tables, si les données sont contenues dans le fichier de « dump ». (Il existe une option pour exporter seulement les schémas des tables.) Ci-contre à droite se trouve un exemple d'une des sections de création de table.

La première chose dont nous devons nous débarrasser se trouve dans la dernière ligne. Tout ce qui suit la parenthèse fermante doit disparaître. (SQLite ne supporte pas une base de données InnoDB). De plus, SQLite ne supporte pas la ligne « PRIMARY KEY ». Dans SQLite, on règle une clé primaire en utilisant « INTEGER PRIMARY KEY AUTOINCREMENT » quand nous définissons la colonne. L'autre chose que SQLite ne supporte pas est le mot-clé « unsigned ».

Quant aux données, les déclarations « INSERT INTO » sont également non-compatibles. Le problème ici est que SQLite ne permet pas les insertions multiples dans une même déclaration. Voici un court exemple tiré du fichier de « dump » MySQL. Remarquez (à droite) que le marqueur de fin de ligne est un point-virgule.

Nous allons également ignorer toutes les lignes de commentaires et

```
DROP TABLE IF EXISTS `categoriesmain`;
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL
auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES
 (1,'Apéritif'),
 (2,'Snack'),
 (3,'Barbecue'),
 (4,'Gateaux'),
 (5,'Bonbons'),
 (6,'Boissons');
```

Pour rendre ceci compatible, nous devons le remplacer par plusieurs insertions séparées, comme ceci :

```
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (1,'Apéritif');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (2,'Snack');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (3,'Barbecue');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (4,'Gateaux');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (5,'Bonbons');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (6,'Boissons');
```

les instructions « CREATE DATABASE » et « USE ». Une fois que nous aurons le fichier SQL converti, nous utiliserons un programme semblable à SQLite Database Browser qui est dans le domaine public, pour réellement créer la base de données, les tables et les données.

Commençons. Ouvrez un dossier pour ce nouveau projet et un nouveau fichier python. Nommez-le MonSQLversSQLite.py.

Vous voyez en haut à droite la déclaration d'importation, la définition de classe et la routine __init__.

Ce programme sera exécuté en ligne de commande, nous avons donc besoin de créer la déclaration « if __name__ », un gestionnaire pour les arguments de ligne de commande et une routine d'utilisation (si l'utilisateur ne sait pas comment utiliser le programme). Tout cela va à la toute fin du programme. Tout le reste du code se trouvera avant ceci :

```
def error(message):
    print >> sys.stderr,
str(message)
```

Ensuite se trouve le gestionnaire qui affiche les instructions d'utilisation du programme.

La routine FaitLe() est appelée si notre programme est lancé à partir de la ligne de commande, ce pourquoi il est conçu. Cependant, si nous voulons pouvoir en faire une bibliothèque qui sera incluse dans un autre programme à un autre moment, nous pouvons simplement utiliser la classe. Ici nous avons mis en place un certain nombre de variables pour s'assurer que tout fonctionne correctement. Le code visible sur la page précédente

```
def FaitLe():
    #=====================================
    #            Variables
    #=====================================
    FichierSource = ''
    FichierDestination = ''
    Debug = False
    Aide = False
    SchemaSeulement = False
    #=====================================
```

```
    #!/usr/bin/env python
    #=====================================
    # MonSQLversSQLite.py
    #=====================================
    #            IMPORTS
    import sys
    #=====================================

    #=====================================
    #     DEBUT CLASS MonSQL2versSQLite
    #=====================================
    class MonSQLversSQLite:
        def __init__(self):
            self.FichierSource = ""
            self.FichierDestination = ""
            self.EcrireFichier = 0
            self.ModeDebug = 0
            self.SchemaSeulement = 0
            self.ModeDirect = False
```

```
if len(sys.argv) == 1:
    usage()
else:
    for a in sys.argv:
        print a
        if a.startswith("FicEntree="):
            pos = a.find("=")
            FichierSource = a[pos+1:]
        elif a.startswith("FicSortie="):
            pos = a.find("=")
            FichierDestination = a[pos+1:]
        elif a == 'Debug':
            Debug = True
        elif a == 'SchemaSeukelent':
            SchemaSeulement = True
        elif a == '-Aide' or a == '-H' or a == '-?':
            Aide = True
    if Aide == True:
        usage()
    r = MonSQLversSQLite()
    r.Init(FichierSource,FichierDest,Debug,SchemaSeul)
    r.ExecuterTravail()
```

en bas à droite analyse ensuite les arguments de ligne de commande passés à notre programme et prépare les choses pour les routines principales.

Quand nous commençons le programme, nous devons fournir au moins deux variables sur la ligne de commande. Ce sont les fichiers d'entrée et de sortie. Nous fournirons également une information pour permettre à l'utilisateur de voir ce qui se passe pendant que le programme est lancé, une option pour simplement créer les tables, ne pas charger les données et un moyen pour l'utilisateur d'appeler au secours. La ligne de commande « normale » pour démarrer le programme ressemble à ceci :

```
MonSQLversSQLite FicEntree=Foo
FicSortie=Bar
```

où « Foo » est le nom du fichier de dump MySQL, et « Bar » est le nom du fichier SQLite que le programme doit créer.

Vous pouvez également l'appeler ainsi :

```
MonSQLversSQLite FicEntree=Foo
FicSortie=Bar Debug SchemaSeu-
lement
```

```
def usage():
    message = (
       '================================================================\n'
        'MonSQLversSQLite - Un convertisseur de bases de donnees\n'
        'Auteur: Greg Walters\n'
        'USAGE:\n'
        'MonSQLversSQLite FicEntree=nomFichier [FicSortie=nomFichier] [SchemaSeulement]
[Debug] [-H-Aide-?]\n'
        '   avec\n'
        '         FicEntree est le fichier de dump MySQL\n'
        '         FicSortie (optionnel) est le nom du fichier de sortie\n'
        '            (si FicSortie est omis, on suppose une sortie directe vers SQLite)\n'
        '         SchemaSeulement (optionnel) Cree les tables, N\'IMPORTE PAS LES
DONNEES\n'
        '         Debug (optionnel) - Affiche les messages de debug\n'
        '         -H or -Aide or -? - Affiche ce message\n'
        'Copyright (C) 2011 par G.D. Walters\n'
        '================================================================\n'
        )
    error(message)
    sys.exit(1)

if __name__ == "__main__":
    FaitLe()
```

ce qui ajoutera l'option pour afficher les messages de débogage et pour créer SEULEMENT les tables sans importer les données.

Finalement si l'utilisateur demande de l'aide, on va simplement dans la section « Utilisation » du programme.

Avant de continuer, regardons à nouveau comment fonctionne la prise en charge des arguments de la ligne de commande.

Lorsqu'un utilisateur entre le nom du programme en ligne de commande (terminal), le système d'exploitation conserve la trace des informations saisies et il les passe au programme juste au cas où des options ont été saisies. Si aucune option (autrement nommée argument) n'est saisie, le nombre d'arguments est un, ce qui correspond au nom de l'application - dans notre cas, MonSQLversSQLite.py. On accède à ces arguments avec la commande sys.arg. Si le nombre est supérieur à un, nous allons utiliser une boucle for pour y accéder. Nous allons parcourir la liste des arguments et vérifier chacun d'eux. Certains programmes exigent que vous entriez les arguments dans un ordre précis. En utilisant l'approche avec une boucle for, les arguments peuvent être saisis dans n'importe quel ordre. Si l'utilisateur ne fournit pas d'argument, ou utilise l'un des arguments d'aide, on affiche l'écran d'utilisation. Ci-dessus se trouve la routine pour cela.

Ensuite, une fois que nous avons analysé l'ensemble des arguments, nous instancions la classe, appelons la rou-

tine de configuration qui remplit certaines variables et ensuite appelons la routine ExecuterTravail. Nous allons commencer notre classe maintenant (voir en bas à droite).

Voici (en haut à droite) les configurations et la routine __init__. Ici, nous configurons les variables dont nous aurons besoin tout au long du code. Souvenez-vous que juste avant d'appeler la routine ExecuterTravail, nous appelons la routine de configuration, où nous prendrons les variables vides pour leur assigner des valeurs correctes. Notez qu'on laisse la possibilité de ne pas écrire dans un fichier, ce qui est utile pour le débogage. Nous avons également la possibilité de simplement écrire le schéma (la structure de la base de données), sans écrire les données. Ceci est utile si vous prenez une base de données et commencez un nouveau projet sans vouloir utiliser des données existantes.

Nous commençons par ouvrir le fichier de « dump » SQL, puis nous définissons les variables à portée interne. Nous définissons aussi certaines chaînes pour nous éviter de les saisir plus tard.

```
    def Init(self, Entree, Sortie = '', Debug = False, Schema =
0):
        self.FichierSource = Entree
        if Sortie == '':
            self.EcrireFichier = 0
        else:
            self.EcrireFichier = 1
            self.FichierDestination = Sortie
        if Debug == True:
            self.ModeDebug = 1
        if Schema == 1:
            self.SchemaSeulement = 1
```

Maintenant passons à la routine ExecuterTravail, où la magie opère vraiment.

```
    def ExecuterTravail(self):
        f = open(self.FichierSource)
        print "Debut du processus"
        cntr = 0
        ModeInsertion = 0
        ModeCreationTable = 0
        DebutInsertion = "INSERT INTO "
        AI = "auto_increment"
        PK = "PRIMARY KEY "
        IPK = " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL"
        CT = "CREATE TABLE "
        # Debut
        if self.EcrireFichier == 1:
            FichierDest = open(self.FichierDestination,'w')
```

```
    while 1:
        ligne = f.readline()
        cntr += 1
        if not ligne:
            break
        # Ignore les lignes vides, ou qui commencent
par "--", ou les commentaires (/*!)
        if ligne.startswith("--"): # Commentaire
            pass
        elif len(ligne) == 1: # Ligne vide
            pass
        elif ligne.startswith("/*!"): # Commentaire
            pass
        elif ligne.startswith("USE"):
            #Ignore les lignes USE
            pass
        elif ligne.startswith("CREATE DATABASE "):
            pass
```

```
#====================================
#     DEBUT CLASS MonSQLversSQLite
#====================================
class MonSQLversSQLite:
    def __init__(self):
        self.FichierSource = ""
        self.FichierDestination = ""
        self.EcrireFichier = 0
        self.ModeDebug = 0
        self.SchemaSeulement = 0
```

Ensuite, si nous prévoyons d'écrire un fichier de sortie, nous l'ouvrons puis nous commençons le processus complet. Nous lirons chaque ligne du fichier d'entrée, pour les traiter et, éventuellement, les écrire dans le fichier de sortie. Nous utilisons une boucle while infinie pour la lecture des lignes, avec une commande « break » quand il ne reste rien dans le fichier d'entrée. Nous utilisons f.readline() pour obtenir la ligne à convertir et nous l'assignons à la variable « ligne ». Certaines lignes peuvent être ignorées. Nous allons simplement utiliser une instruction if/elif suivie par une instruction pass pour cela (page précédente en bas à gauche).

Ensuite nous pouvons cesser d'ignorer les choses et agir pour de bon. Si nous rencontrons une déclaration de création de table, nous allons commencer ce processus. Rappelez-vous, nous avons défini CT comme étant égal à “CREATE TABLE”. Ici (en haut à droite), nous réglons une variable « ModeCreationTable » sur 1, pour savoir que c'est ce que nous faisons, car chaque définition de champ est sur une ligne distincte. Nous prenons ensuite notre ligne, supprimons le retour chariot, la préparons pour être écrite dans le fichier de sortie, et, si nécessaire, nous l'écrivons.

Maintenant (à droite au milieu), nous devons commencer à traiter chaque ligne contenue dans l'instruction de création de table - en manipulant chaque ligne pour que SQLite soit content. Il y a plusieurs choses que SQLite ne traitera pas. Regardons à nouveau une instruction CREATE TABLE de MySQL.

Une chose qui va vraiment poser problème à SQLite est la toute dernière ligne après la parenthèse fermante. Une autre est la ligne juste au-dessus, la ligne de clé primaire. Une autre chose est le mot-clé unsigned à la deuxième ligne. Cela va nécessiter un peu de code (en bas à gauche) pour contourner ces problèmes, mais nous pouvons y arriver.

Tout d'abord, (troisième cadre sur la droite), nous vérifions si la ligne commence par « auto_increment » (AI). Nous supposerons que ce sera la ligne de clé primaire. Bien que cela soit vrai 98,6 % du temps, ce n'est pas toujours le cas.

```
    elif ligne.startswith(CT):
        ModeCreationTable = 1
        l1 = len(ligne)
        ligne = ligne[:l1-1]
        if self.ModeDebug == 1:
            print "Debut de CREATE TABLE"
            print ligne
        if self.EcrireFichier == 1:
            FichierDest.write(ligne)
```

```
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
            p1 = ligne.find(AI)
            if ligne.startswith(") "):
                ModeCreationTable = 0
                if self.ModeDebug == 1:
                    print "Fin du CREATE TABLE"
                nouvelleLigne = ");\n"
                if self.EcrireFichier == 1:
                    FichierDest.write(nouvelleLigne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(nouvelleLigne)
```

```
            elif p1 != -1:
                # on a une ligne de cle primaire
                l = ligne.strip()
                fnpos = l.find(" int(")
                if fnpos != -1:
                    fn = l[:fnpos]
                nouvelleLigne = fn + IPK #+ ",\n"
                if self.EcrireFichier == 1:
                    FichierDest.write(nouvelleLigne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(nouvelleLigne)
```

```
    elif ModeCreationTable == 1:
        # traite la ligne...
        if self.ModeDebug == 1:
            print "Ligne a traiter — {0}".format(ligne)
```

Cependant, nous allons garder les choses simples. Ensuite nous vérifions si la ligne commence par «) ». Cela signifie que ceci est la dernière ligne de la section CREATE TABLE. Si oui, nous écrivons simplement une chaîne pour fermer correctement la déclaration dans la variable « nouvelleLigne », réglons la variable ModeCreationTable à 0 et, si nous écrivons dans un fichier, nous réalisons l'écriture.

Maintenant (page précédente en bas à droite), nous utilisons les informations que nous avons trouvées sur le mot-clé auto_increment. Tout d'abord, nous enlevons de la ligne tous les espaces parasites, puis vérifions pour voir où se trouve (nous supposons qu'elle est là) l'expression « int(» dans la ligne. Nous la remplacerons par l'expression « INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ». La longueur de l'entier n'est pas importante pour SQLite. Encore une fois, nous l'écrivons si c'est nécessaire.

Maintenant nous cherchons l'expression « PRIMARY KEY » dans la ligne. Remarquez l'espace supplémentaire à la fin : c'est exprès. Si on la trouve, on ignore la ligne.

```
elif
ligne.strip().startswith(PK):
    pass
```

Maintenant (en haut à droite) nous recherchons l'expression « unsigned » (encore une fois remarquez les espaces supplémentaires) et la remplaçons par « ».

C'est la fin de la routine de création de table. Maintenant (ci-dessous), nous passons aux requêtes d'insertion pour les données. La variable DebutInsertion contient l'expression « INSERT INTO ». Nous vérifions cela parce que MySQL permet d'insérer plusieurs déclarations en une seule commande, mais pas SQLite. Nous devons faire des déclarations distinctes pour chaque bloc de données. Nous réglons une va-

```
        elif ligne.startswith(DebutInsertion):
            if ModeInsertion == 0:
                ModeInsertion = 1
                # recupere le nom de la table et la liste
des champs
                istatement = ligne
                # enleve le retour chariot de la ligne
istatement
                    l = len(istatement)
                    istatement = istatement[:l-2]
```

```
            elif ligne.find(" unsigned ") != -1:
                ligne = ligne.replace(" unsigned "," ")
                ligne = ligne.strip()
                longueur1 = len(ligne)
                ligne = ligne[:l1-1]
                if self.EcrireFichier == 1:
                    FichierDest.write("," + ligne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(ligne)
```

Sinon, on peut s'occuper de la ligne.

```
            else:
                longueur1 = len(ligne)
                ligne = ligne.strip()
                ligne = ligne[:l1-4]
                if self.ModeDebug == 1:
                    print "," + ligne
                if self.EcrireFichier == 1:
                    FichierDest.write("," + ligne)
```

```
                if posx != -1:
                    longueur1 = ligne[:posx+3]
                    ModeInsertion  = 0
                    if self.ModeDebug == 1:
                        print istatement + longueur1
                        print "--------------------------"
                    if self.EcrireFichier == 1:
                        FichierDest.write(istatement +
longueur1+"\n")
```

Sinon, on concatène le prélude à l'instruction et on termine par un point-virgule.

```
                elif pos1 != -1:
                    l1 = line[:pos1+2]
                    if self.DebugMode == 1:
                        print istatement + l1 + ";"
                    if self.WriteFile == 1:
                        OutFile.write(istatement + l1 +
";\n")
```

riable appelée « ModeInsertion » à 1, plaçons le « INSERT INTO {table} {liste des champs} VALUES (» dans une variable réutilisable (que je vais appeler notre prélude), et continuons.

Maintenant, nous vérifions si nous devons seulement travailler sur le schéma. Si oui, nous pouvons ignorer sans problème toutes les instructions d'insertion. Sinon, nous devons nous en occuper.

```
elif self.SchemaSeulement == 0:
    if ModeInsertion == 1:
```

Nous vérifions s'il y a soit « '); » soit « '), » dans notre ligne. Le cas « '); » indique que c'est la dernière ligne de l'ensemble d'instructions d'insertion.

```
posx = ligne.find("');")
pos1 = ligne.find("'),")
longueur1 = ligne[:pos1]
```

Cette ligne vérifie s'il y a des apostrophes échappées et les remplace.

```
ligne = ligne.replace
("\\'","''")
```

Si nous avons une déclaration de clôture (");"), c'est alors la fin de notre ensemble d'insertions et nous pouvons créer l'instruction en concaténant le prélude à l'instruction de valeur proprement dite. Ceci est illustré en bas à droite de la page précédente.

Tout cela fonctionne (ci-dessous) si la dernière valeur que nous avons dans l'instruction INSERT est une chaîne entre guillemets. Cependant, si la dernière valeur est une valeur numérique, nous devons procéder un peu différemment. Vous serez en mesure de comprendre ce que nous faisons ici.

Enfin, nous fermons notre fichier d'entrée et, si nous écrivons un fichier de sortie, nous le fermons aussi.

```
f.close()
if self.EcrireFichier == 1:
    FichierDest.close()
```

Une fois que vous avez votre fichier converti, vous pouvez utiliser SQLite Database Browser pour remplir la structure de la base et les données.

Ce code devrait fonctionner tel quel dans plus de 90 % des cas. Nous pourrions avoir oublié certaines choses à cause d'autres problèmes, c'est pour cela qu'un mode Debug est prévu. Cependant, j'ai testé cela sur plusieurs fichiers et n'ai eu aucun problème.

Comme toujours, le code est disponible sur Pastebin : http://pastebin.com/Bdt64VqS.

À la prochaine fois.

```
else:
    if self.ModeDebug == 1:
        print "Test de la ligne {0}".format(ligne)
    pos1 = ligne.find("),")
    posx = ligne.find(");")
    if self.ModeDebug == 1:
        print "pos1 = {0}, posx = {1}".format(pos1,posx)
    if pos1 != -1:
        longueur1 = ligne[:pos1+1]
        if self.ModeDebug == 1:
            print istatement + longueur1 + ";"
        if self.EcrireFichier == 1:
            FichierDest.write(istatement + longueur1 + ";\n")
    else:
        ModeInsertion = 0
        longueur1 = ligne[:posx+1]
        if self.ModeDebug == 1:
            print istatement + longueur1 + ";"
        if self.EcrireFichier == 1:
            FichierDest.write(istatement + longueur1 + ";\n")
```

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.com.

TUTORIEL

Greg Walters

Débuter Python - Partie 30

Ce mois-ci, nous allons explorer encore un autre concepteur graphique, cette fois c'est pour Tkinter. Beaucoup de gens ont un problème avec Tkinter, car il n'offre pas un designer intégré. Alors que je vous ai montré comment concevoir facilement vos applications sans concepteur, nous allons en examiner un maintenant. Il s'appelle Page. Fondamentalement, il s'agit d'une version de Visual TCL avec une couche de Python par dessus. La version actuelle est la 3.2 et peut être trouvée à http://sourceforge.net/projects/page/files/latest/download.

Pré-requis

Vous devez avoir TCK/TK 8.5.4 ou plus, Python 2.6 ou plus et pyttk, que vous pouvez obtenir (si vous ne l'avez pas encore) à partir de http://pypi.python.org/pypi/pyttk. Vous avez probablement tous ceux-ci à l'exception possible de pyttk.

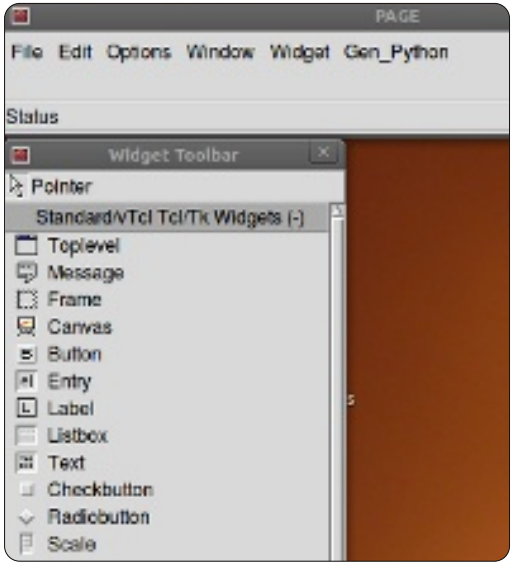
Installation

Vous ne pouvez vraiment pas demander une procédure d'installation plus facile. Il suffit de décompresser le fichier de distribution dans un dossier de votre choix. Exécutez le script appelé « configure » à partir du dossier où vous venez de tout déballer. Cela va créer votre script de lancement appelé « page » que vous utiliserez pour obtenir tout le reste. C'est tout.

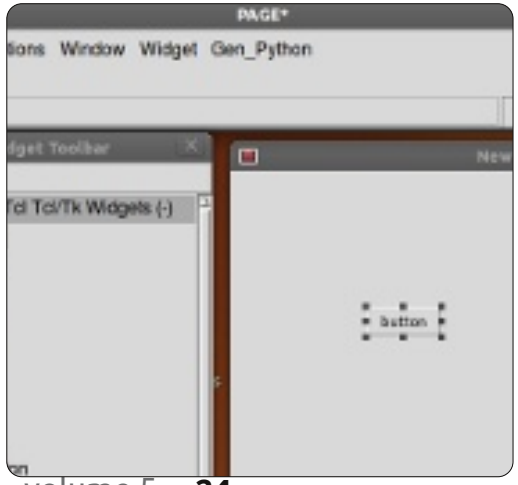
Apprentissage de page

Lorsque vous démarrez Page, vous aurez trois fenêtres (formulaires). L'une est une « piste de lancement », l'autre est une boîte à outils et la dernière montre l'éditeur d'attributs.

Pour démarrer un nouveau projet, cliquez sur le bouton du haut dans la boîte à outils.



Cela crée votre formulaire principal. Vous pouvez le déplacer où vous le souhaitez sur votre écran. Ensuite, et à partir de maintenant, cliquez sur un widget dans la boîte à outils, puis cliquez sur l'endroit où vous le voulez sur le formulaire principal.

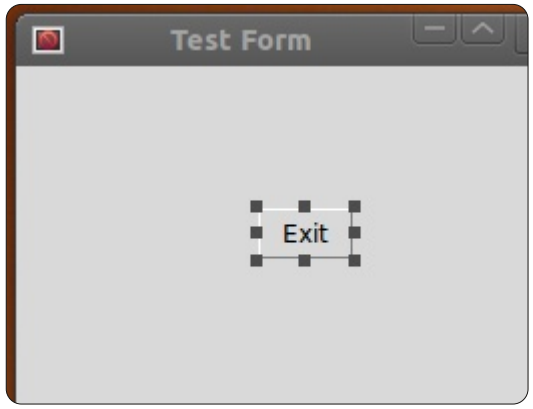


Pour l'instant, nous allons faire un bouton. Cliquez sur le bouton Button dans la boîte à outils, puis cliquez quelque part sur le formulaire principal.

Ensuite, dans le formulaire de lancement, cliquez sur Fenêtre (Window) et sélectionnez l'Éditeur d'attributs (s'il n'est pas déjà affiché). Votre bouton unique devrait être déjà mis en surbrillance ; déplacez-le dans le formulaire et lorsque vous relâchez le bouton de la souris, vous devrez voir le changement de position dans le formulaire éditeur d'attributs sous « x position » et « y position ».

Ici, nous pouvons définir d'autres attributs tels que le texte sur le bouton (ou la plupart des autres widgets), l'alias pour le widget (le nom auquel nous allons nous référer dans notre code), la couleur, le nom par lequel nous l'appellerons et plus. Près du bas de l'éditeur d'attributs se trouve le champ de texte. Ceci est le texte que l'utilisateur voit pour, dans ce cas, le widget bouton. Changeons-le de « Button » à « Exit ». Remarquez qu'à présent le bouton affiche « Exit ». Maintenant redimensionnez le formu-

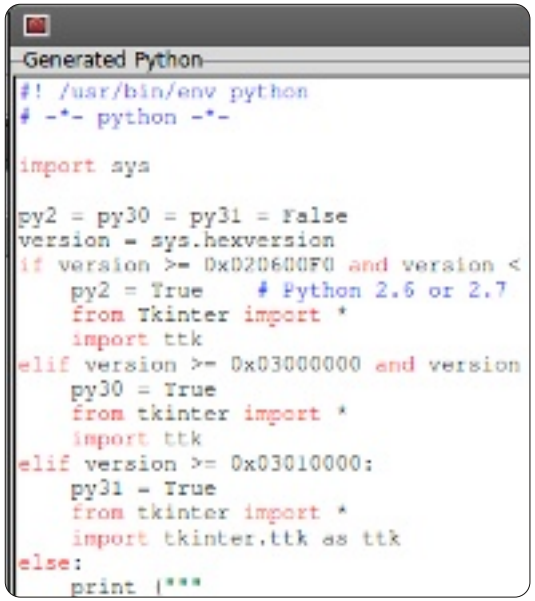
laire pour montrer seulement le bouton et recentrez le bouton dans le formulaire.



Ensuite, cliquez quelque part dans le formulaire principal où le bouton n'est pas. Le formulaire éditeur d'attributs affiche maintenant les attributs du formulaire principal. Trouvez le champ « Title » et changez-le de « New Toplevel 1 » à « Test Form ».

Maintenant, avant de sauvegarder notre projet, nous avons besoin de créer un dossier pour contenir nos fichiers de projet. Créez un dossier quelque part sur votre disque appelé « PageProjects ». Puis, dans la fenêtre de lancement, sélectionnez File puis Save As. Allez dans votre dossier PageProjects et, dans la boîte de dialogue, tapez TestForm.tclet et cliquez sur le bouton Save. Notez que c'est enregistré comme fichier TCL, pas comme fichier Python. Ensuite, nous allons créer le fichier python.

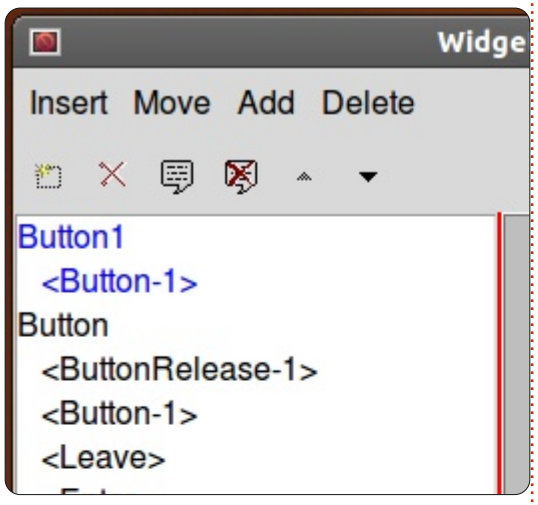
Dans la fenêtre de lancement, cherchez le menu Gen_Python et



cliquez dessus. Sélectionnez Generate Python et un nouveau formulaire apparaît.

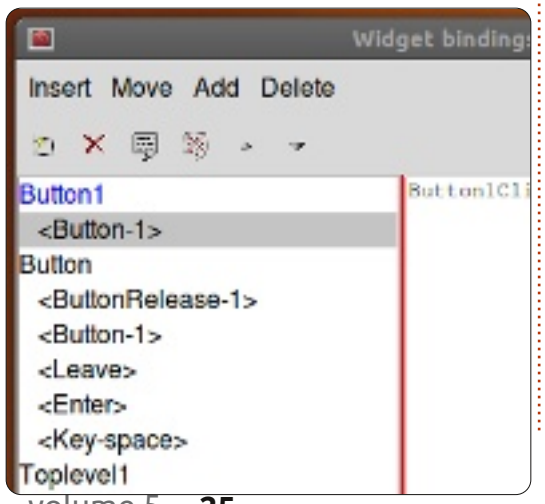
Page a généré (comme son nom l'indique) le code Python à notre place et l'a placé dans une fenêtre pour qu'on puisse le voir. Au bas de ce formulaire, il y a trois boutons : Save, Run et Close.

Cliquez sur Save. Si, à ce stade, vous deviez regarder dans votre dossier PageProjects, vous verriez le fichier python (TestForm.py). Maintenant, cliquez sur le bouton Run. En quelques secondes, vous verrez le projet démarrer. Le bouton n'est pas connecté à quoi que ce soit encore, il ne fera donc rien si vous cliquez dessus. Il suffit de fermer le formulaire avec le



« X » dans le coin de la fenêtre. Maintenant, fermez la fenêtre de console Python avec le bouton de fermeture en bas à droite.

De retour à notre formulaire principal, sélectionnez le bouton Exit et



faites un clic droit dessus. Sélectionnez « Bindings… ». Dans le menu se trouve un ensemble de boutons.

Le premier sur la gauche vous permet de créer une nouvelle liaison. Cliquez sur « Bouton-1 ». Cela nous permet de paramétrer la liaison pour le bouton gauche de la souris. Dans la fenêtre sur la droite, tapez « Button1Click ».

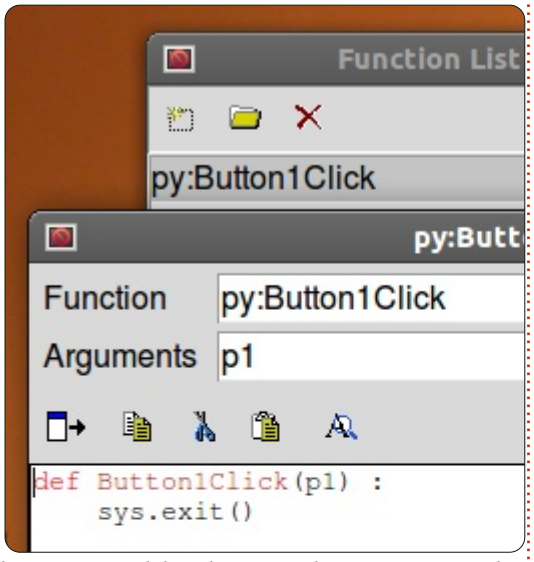
Enregistrez et générez le code python à nouveau. Faites défiler le code dans la console Python jusqu'à la fin du fichier. Au-dessus du code de la « class Test_Form » se trouve la fonction dont nous venons de demander la création. Notez qu'à ce stade, il est simplement transmis. Regardez plus loin vers le bas et vous

verrez le code qui crée et contrôle notre bouton. Tout est fait pour nous déjà. Toutefois, nous devons encore dire au bouton ce qu'il faut faire. Fermez la console Python et continuons.

Sur la fenêtre de lancement, cli-

quez sur Window puis sélectionnez Function List. Ici, nous allons écrire notre méthode pour fermer la fenêtre.

Le premier bouton à gauche est le



bouton Add. Cliquez dessus. Dans la zone Function, tapez « py:Button1Click » et, dans la zone Arguments, tapez « 1 » et modifiez le texte dans la zone inférieure à :

```
def Button1Click(p1):
        sys.exit()
```

Cliquez sur la coche et nous avons terminé avec cela.

Ensuite, nous devons lier cette routine au bouton. Sélectionnez le bouton dans le formulaire, faites un clic droit dessus et sélectionnez « Bindings… ». Comme précédemment, cliquez sur le bouton le plus à gauche sur la barre d'outils et sélectionnez le Button-1. C'est l'événement correspondant au clic gauche de la souris. Dans la boîte de texte à droite, entrez « Button1Click ». Assurez-vous d'utiliser la même casse que pour la fonction que nous venons de créer. Cliquez sur la coche sur le côté droit. Maintenant, sauvegardez et générez votre code python.

Vous devriez voir le code suivant vers le bas, mais en dehors de la classe Test_Form :

```
def Button1Click(p1) :

sys.exit()
```

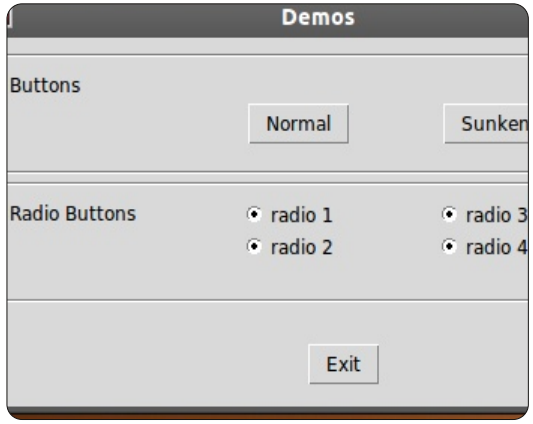
Et la dernière ligne de la classe devrait être :

```
self.Button1.bind('<Button-1>',Button1Click)
```

Maintenant, si vous exécutez votre code et cliquez sur le bouton Exit, le formulaire doit se fermer correctement.

Pour aller plus loin

Maintenant, nous allons faire quelque chose de plus compliqué. Nous allons créer une démo montrant quelques-uns des widgets qui sont disponibles. D'abord fermez Page et redémarrez-le. Ensuite, créez un nouveau formulaire Toplevel. Ajoutez deux cadres, l'un au-dessus de l'autre et étirez-les pour prendre à peu près toute la largeur du formulaire. Dans le cadre du haut, placez une étiquette de texte et, en utilisant l'éditeur attributs, modifiez le texte à « Buttons: ». Ensuite, ajoutez deux boutons dans le plan horizontal. Modifiez le texte de celui de gauche en « Normal » et celui de droite en « Sunken » [Ndt : en creux]. Alors que le bouton Sunken est



sélectionné, changez le relief de « Sunken » et nommez-le btnSunken. Nommez le bouton « Normal » « btnNormal ». Enregistrez ce projet avec le nom « Demos.tcl ».

Ensuite, placez dans le cadre inférieur une étiquette de texte « Radio Buttons » et quatre boutons radio comme dans l'image ci-dessous. Enfin, placez un bouton Exit en dessous du cadre inférieur.

Avant de travailler sur les liaisons, nous allons créer nos fonctions de clic. Ouvrez la liste de fonctions et créez deux fonctions. Le premier devrait être appelé btnNormalClicked et l'autre btnSunkenClicked. Assurez-vous d'inclure p1 dans la boîte d'arguments. Voici le code que vous devez avoir pour eux :

```
def btnNormalClicked(p1):

print "Normal Button Clicked"

def btnSunkenClicked(p1) :

print "Sunken Button Clicked"
```

Ajoutons nos liaisons aux boutons. Pour chaque bouton, faites un clic droit dessus, sélectionnez « Bindings… » et ajoutez, comme avant, une liaison aux fonctions que nous avons créées. Pour le bouton normal, cela sera « btnNormalClicked » et pour le bouton creux cela sera « btnSunkenClicked ». Enregistrez et générez votre code. Maintenant, si vous testiez le programme avec l'option « Run » de la console Python et cliquiez sur un des boutons, vous ne verriez rien se produire. Toutefois, lorsque vous fermerez l'application, vous devriez voir les réponses écrites. Ceci est normal pour Page et si vous l'aviez simplement exécuté à partir de la ligne de commande comme vous le feriez nor-

malement, les choses devraient fonctionner comme prévu.

Maintenant, passons à nos boutons radio. Nous les avons regroupés en deux groupes « clusters ». Les deux premiers (Radio 1 et Radio 2) formeront le groupe 1 et les deux autres seront le groupe 2. Cliquez sur Radio 1 et dans l'éditeur d'attributs, définissez la valeur à 0 et la variable à « rbc1 ». Définissez la variable pour Radio 2 à « rbc1 » et la valeur à 1. Faites la même chose pour Radio 3 et Radio 4, mais pour les deux réglez la variable à « rbc2 ». Si vous voulez, vous pouvez améliorer le clic des boutons radio et imprimer quelque chose dans le terminal, mais pour l'instant, la chose importante est que les groupes fonctionnent. Cliquer sur Radio1 désélectionnera Radio2 et n'aura pas d'influence sur Radio3 ou Radio4, et de même pour Radio2 et ainsi de suite.

Enfin, vous devez créer une fonction pour le bouton Exit et la lier au bouton, comme nous l'avons fait dans le premier exemple.

Si vous avez suivi depuis le début nos autres applications Tkinter, vous devriez être en mesure de comprendre le code montré ci-dessus à droite. Sinon, merci de retourner dans quelques numéros précédents pour lire une présentation complète de ce code.

```
def set_Tk_var():
# These are Tk variables passed to Tkinter and must
# be defined before the widgets using them are created.
global rbc1
rbc1 = StringVar()
global rbc2
rbc2 = StringVar()
def btnExitClicked(p1) :
sys.exit()
def btnNormalClicked(p1) :
print "Normal Button Clicked"
def btnSunkenClicked(p1) :
print "Sunken Button Clicked"
```

Vous pouvez voir qu'utiliser Page rend le processus de conception de base beaucoup plus facile que de le faire vous-même. Nous avons seulement commencé à examiner ce que peut faire Page et nous allons faire quelque chose de beaucoup plus réaliste la prochaine fois.

Le code python peut être trouvé sur pastebin à : http://pastebin.com/ts3MKyCZ.

Une note avant de terminer pour ce mois-ci. Vous avez sans doute remarqué que j'ai manqué quelques numéros. Cela est dû au fait que ma femme a été diagnostiquée avec un cancer l'an dernier. Même si j'ai vraiment essayé d'empêcher les choses de tomber à travers les mailles du filet, un certain nombre de choses y sont passées. Une de ces choses est mon ancien domaine/site web www.thedesignatedgeek.com. J'ai fait une grossière erreur et en ai raté le renouvellement. Pour cette raison, le domaine a été vendu sans mon consentement. J'ai mis en place www.thedesignatedgeek.net avec tous les vieux trucs. Je vais travailler dur le mois prochain pour mettre tout cela à jour.

Rendez-vous la prochaine fois.

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.com.

ubuntu uk podcast

Le Podcast Ubuntu couvre toutes les dernières nouvelles et les problèmes auxquels sont confrontés les utilisateurs de Linux Ubuntu et les fans du logiciel libre en général. La séance s'adresse aussi bien au nouvel utilisateur qu'au plus ancien codeur. Nos discussions portent sur le développement d'Ubuntu, mais ne sont pas trop techniques. Nous avons la chance d'avoir quelques supers invités, qui viennent nous parler directement des derniers développements passionnants sur lesquels ils travaillent, de telle façon que nous pouvons tous comprendre ! Nous parlons aussi de la communauté Ubuntu et de son actualité.

Le podcast est présenté par des membres de la communauté Ubuntu Linux du Royaume-Uni. Il est couvert par le Code de Conduite Ubuntu et est donc adapté à tous.

L'émission est diffusée en direct un mardi soir sur deux (heure anglaise) et est disponible au téléchargement le jour suivant.

podcast.ubuntu-uk.org

TUTORIEL

Débuter Python - Partie 31

Greg Walters

Après notre dernière rencontre, vous devriez avoir une assez bonne idée de la façon d'utiliser Page. Sinon, allez vite lire l'article du mois dernier. Nous allons continuer cette fois-ci en créant une application de liste de fichiers avec une interface graphique. Le but ici est de créer une application graphique qui va récursivement parcourir un répertoire, en cherchant des fichiers avec un ensemble défini d'extensions, et afficher le résultat dans une vue arborescente. Pour cet exemple, nous allons chercher les fichiers multimédias avec les extensions .avi, .mkv, .mv4, .mp3 et .ogg.

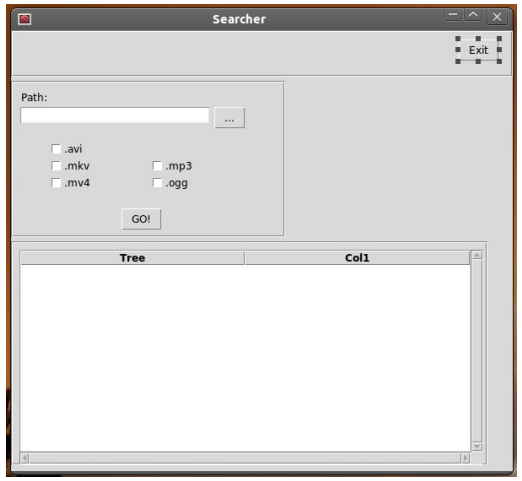
Cette fois, le texte peut sembler un peu laconique dans la partie conception. Tout ce que je vais faire, c'est vous donner des indications pour le placement des widgets, ainsi que les attributs et les valeurs requises, de cette façon :

```
widget

attribut: valeur
```

Je ne citerai les chaînes de texte que lorsque cela sera nécessaire. Par exemple, pour l'un des boutons, le texte doit être réglé sur « … ».

Voici à quoi va ressembler l'interface graphique de notre application :



Comme vous pouvez le voir, nous avons notre formulaire principal, un bouton pour quitter, une boîte de saisie de texte avec un bouton qui va appeler une boîte de dialogue pour demander le répertoire, cinq cases à cocher pour sélectionner les types d'extension, un bouton « ALLER ! » pour effectivement commencer le traitement et une arborescence pour afficher notre production.

Nous pouvons commencer. Lancez Page et créez un nouveau widget principal. En utilisant l'éditeur d'attributs, définissez les attributs suivants :

```
alias: rechercher
titre: rechercher
```

Assurez-vous de sauvegarder souvent. Lorsque vous enregistrez le fichier, donnez-lui le nom « Rechercher ». Rappelez-vous, Page ajoute l'extension .tcl à votre place et quand vous générerez le code python il sera sauvegardé dans le même dossier.

Ensuite, ajoutez un cadre. Il devrait se placer tout en haut du cadre principal. Définissez les attributs comme suit :

```
Largeur : 595
Hauteur : 55
Position x : 0
Position y : 0
```

Dans ce cadre, ajoutez un bouton. Ce sera notre bouton Quitter.

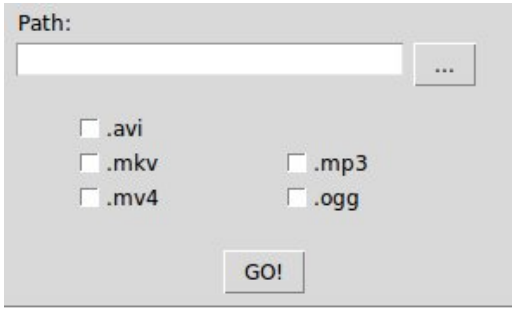
```
Alias : btnQuitter
Texte : Quitter
```

Déplacez-le au centre de la fenêtre, ou alors sur le côté droit. J'ai mis le mien à X = 530 et Y = 10.

Créez un autre cadre :

```
Largeur : 325
Hauteur : 185
Position y : 60
```

Voici à quoi ce cadre va ressembler, pour vous guider à travers cette section (colonne suivante).



Dans ce cadre, ajoutez une étiquette. Définissez l'attribut texte à « Chemin : ». Déplacez-le en haut à gauche de la fenêtre.

Dans le même cadre, ajoutez un widget de saisie :

```
Alias : txt Chemin
Texte : CheminFichier
Largeur : 266
Hauteur : 21
```

Ajoutez un bouton à droite de la zone de saisie :

```
Alias : btnCheminRecherche
Texte : « … » (sans guillemets).
```

Ajoutez cinq (5) cases à cocher. Mettez-les dans l'ordre suivant :

```
x
x   x
x   x
```

Les trois cases à cocher de gauche

sont pour les fichiers vidéo et les deux de droite sont pour les fichiers audio. Nous allons d'abord traiter les trois de gauche, puis les deux de droite.

```
Alias: chkAVI
Texte : ".avi" (sans guillemets)
Variable : VchkAVI

Alias : chkMKV
Texte : ".mkv" (sans guillemets)
Variable : VchkMKV

Alias: chkMV4
Texte : ".mv4"(sans guillemets)
Variable: VchkMV4

Alias : chkMP3
Texte : ".mp3" (sans guillemets)
Variable : VchkMP3

Alias : chkOGG
Texte : ".ogg" (sans guillemets)
Variable : VchkOGG
```

Enfin ajoutez dans ce cadre un bouton quelque part en dessous des cinq cases à cocher et un peu centré à l'intérieur du cadre :

```
Alias : btnAller
Texte : ALLER
```

Maintenant, ajoutez un autre cadre en dessous du précédent :

```
Largeur : 565
Hauteur : 265
```

J'ai placé le mien à environ X = 0 et Y = 250. Vous pourriez avoir à redimensionner votre formulaire principal pour voir l'affichage en entier. Dans ce cadre, ajoutez un widget Scrolledtreeview (vue arborescente avec ascenseur) :

```
Largeur : 550
Hauteur : 254
Position X : 10
Position Y : 10
```

Voilà. Nous avons conçu notre interface graphique. Maintenant tout ce qu'il reste à faire est de créer notre liste de fonctions et de lier ces fonctions à nos boutons.

Dans la fenêtre de liste des fonctions, cliquez sur le bouton Nouveau (le bouton le plus à gauche). Ceci nous amène à l'éditeur de nouvelle fonction. Modifiez le texte dans la zone de saisie Fonction en remplaçant « py:xxx » par « py:btnClicQuitter() ». Dans la zone de texte de l'argument saisissez « p1 ». Dans la zone de saisie multilignes du bas, changez le texte en :

```
def btnClicQuitter(p1):

    sys.exit()
```

Notez que ce n'est pas indenté. Page le fera pour nous quand il créera le fichier python.

Ensuite, créez une autre fonction appelée btnClicAller. N'oubliez pas d'ajouter un paramètre nommé « p1 ». Laissez l'instruction « pass » ; nous changerons cela plus tard.

Enfin, ajoutez une autre fonction appelée « btnCheminRecherche ». Encore une fois, laissez l'instruction « pass ».

En tout dernier lieu, nous devons relier les boutons et les fonctions que nous venons de créer.

Faites un clic droit sur le bouton Quitter que nous avons créé, sélectionnez Lier. Une grande boîte apparaîtra. Cliquez sur le bouton Nouvelle liaison, cliquez sur bouton-1 et remplacez le mot « A FAIRE » dans la boîte de saisie de texte de droite par « btnClicQuitter ». NE METTEZ PAS les parenthèses () ici.

Liez le bouton ALLER à la fonction btnClicAller et le bouton « … » à btnClicCheminRecherche.

Sauvegardez votre interface graphique et générez le code python.

Maintenant tout ce qu'il reste à faire est de créer le code qui « agglutine » l'interface graphique.

Ouvrez le code que nous venons de générer dans votre éditeur de texte favori. Commençons par examiner ce que Page a créé pour nous.

Au début du fichier se trouve l'en-tête standard python et une déclaration d'importation unique pour importer la bibliothèque système (sys). Ensuite vient du code plutôt confus (à première vue). C'est simplement pour examiner la version de python avec laquelle vous essayez d'exécuter l'application, puis pour importer les versions correctes des bibliothèques Tkinter. À moins que vous n'utilisiez Python 3.x, vous pouvez tout simplement ignorer les deux derniers.

Nous allons modifier la portion de code 2.x dans quelques instants pour importer d'autres modules Tkinter.

Arrive ensuite la routine « vp_start_gui() ». C'est la routine principale du programme. Ceci met en place notre interface, définit les variables dont nous avons besoin et appelle ensuite la boucle principale Tkinter. Vous remarquerez peut-être la ligne « w = None » juste en dessous. Elle n'est pas indentée et n'a pas besoin de l'être.

Ensuite viennent deux routines (create_Rechercher et destroy_Rechercher) qui sont utilisées pour remplacer la routine principale si nous utilisons cette application comme une bibliothèque. Nous n'avons pas besoin de nous inquiéter à ce sujet.

Arrive ensuite la routine « initialise_var_Tk ». Nous définissons les variables Tkinter utilisées qui doivent être mises en place avant de créer les widgets. Vous pouvez sans doute reconnaître la variable texte pour le widget de saisie CheminFichier et les variables de nos cases à cocher. Les trois routines suivantes sont les fonctions que nous avons créées en utilisant l'éditeur de fonctions et une fonction « init() ».

Exécutez le programme maintenant. Notez que les cases à cocher contiennent des coches grisées. Nous ne voulons pas cela dans notre application finale, nous allons donc créer un peu de code pour les faire disparaître avant que le formulaire ne soit affiché à l'utilisateur. La seule chose qui fonctionne à part les cases à cocher est le bouton Quitter.

Utilisez-le pour terminer le programme.

Maintenant, nous allons jeter un coup d'oeil à la classe qui contient effectivement la définition de l'interface graphique. Il s'agit de la classe « Chercheur ». C'est là que tous les widgets sont définis et placés dans notre formulaire. Vous devez être familier avec cela maintenant.

Deux classes de plus sont créées, elles contiennent le code pour gérer l'arborescence qui défile. Nous n'avons pas à changer tout cela. Tout a été créé pour nous par Page.

Revenons maintenant au début du code et commençons à le modifier.

Nous avons besoin d'importer quelques modules de bibliothèque de plus ; pour cela ajoutez en dessous de la déclaration « import sys » :

```
import os

from os.path import join,
getsize, exists
```

Maintenant, trouvez la section qui contient la ligne « py2 = True ». Comme nous l'avons dit, c'est la section qui traite des importations tkinter pour Python version 2.x. En dessous de « import ttk », nous avons besoin d'ajouter ce qui suit pour utiliser la bibliothèque FileDialog. Nous avons également besoin d'importer le module tkFont :

```
import tkFileDialog

import tkFont
```

Ensuite nous devons ajouter quelques variables à la routine « initialise_var_Tk() ». En bas de la routine, ajoutez les lignes suivantes :

```
global exts, FileList

exts = []

ListeFichiers=[]
```

Ici, nous créons deux variables globales (exts et ListeFichiers) qui seront utilisées plus tard dans notre code. Les deux sont des listes. « exts » est une liste des extensions que l'utilisateur sélectionne dans l'interface. « ListeFichiers » contient une liste des fichiers correspondants à la recherche effectuée par l'utilisateur. Nous allons l'utiliser pour remplir le widget de vue arborescente.

Puisque notre « btnClicQuitter » est déjà créé pour nous par Page, nous allons nous occuper de la routine « btnClicAller ». Commentez la déclaration pass et ajoutez le code de sorte qu'il ressemble à ceci :

```
def btnClicAller(p1) :

#pass

ConstruireExts()

chemin = CheminFichier.get()

e1 = tuple(exts)

Parcourir(chemin,e1)

ChargerDonnees()
```

C'est la routine qui est appelée lorsque l'utilisateur clique sur le bouton « ALLER ». Nous appelons une routine nommée « ConstruireExts » qui crée la liste des extensions que l'utilisateur a sélectionnée. Puis nous récupérons le chemin que l'utilisateur a choisi dans la boîte de dialogue de demande de répertoire et l'assignons à la variable chemin. Nous créons ensuite un tuple à partir de la liste des extensions, ce qui est nécessaire quand nous vérifions les fichiers. Nous appelons ensuite une routine appelée « Parcourir » en lui passant le répertoire cible et le tuple des extensions.

Enfin, nous appelons une routine nommée « ChargerDonnees ».

Ensuite, nous devons étoffer la routine « btnClicCheminRecherche ». Commentez la déclaration pass et modifiez le code pour qu'il ressemble à ceci :

```
def
btnClicCheminRecherche(p1) :

     #pass

     chemin =
tkFileDialog.askdirectory()
#**self.file_opt)

     CheminFichier.set(chemin)
```

Puis vient la routine init. À nouveau, le code doit ressembler à ceci :

```
def init():

    #pass

    # se lance apres la creation
des fenetres et des widgets...

    global VueArborescente

    InitialiserCases()

    VueArborescente =
w.Scrolledtreeview1


InitialiserVueArborescente()
```

Ici, nous créons une variable globale appelée VueArborescente. Nous appelons ensuite une routine qui efface les contrôles gris dans les cases à cocher, affectons la variable VueArborescente pour pointer vers l'arborescence avec ascenseurs de notre formulaire et appelons

InitialiserVueArborescente pour définir les en-têtes pour les colonnes. Voici le code de la routine InitialiserCases qui doit être la suivante :

```
def InitialiserCases():

    VchkAVI.set('0')

    VchkMKV.set('0')

    VchkMP3.set('0')

    VchkMV4.set('0')

    VchkOGG.set('0')
```

Ici, tout ce que nous faisons, c'est de définir les variables (ce qui définit automatiquement l'état d'activation dans nos cases à cocher) à 0. Si vous vous souvenez, à chaque fois qu'on clique sur la case, cette variable est automatiquement mise à jour. Si la variable est modifiée par notre code, la case à cocher répond également. Maintenant (en haut à droite) nous allons nous occuper de la routine qui établit la liste des extensions à partir de ce que l'utilisateur a cliqué.

Essayez de vous rappeler mon neuvième article dans le FCM nº 35. Nous avons écrit du code pour créer un catalogue de fichiers MP3. Nous allons utiliser une version abrégée de cette routine (au milieu à droite). Reportez-vous au FCM nº 35 si vous avez des questions au sujet de cette routine.

Ensuite (en bas à droite) nous appelons la routine InitialiserVueArborescente. C'est assez simple. Nous définissons une variable « TitresColonnes » avec les rubriques que nous voulons dans chaque colonne de l'arborescence. Nous utilisons une liste pour cela. Nous réglons ensuite l'attribut titre de chaque colonne. Nous réglons également la largeur de colonne à la taille de cet en-tête.

Enfin, nous devons créer la routine « ChargerDonnees » (page suivante, en haut à droite) qui est l'endroit où nous chargeons nos données dans l'arborescence. Chaque ligne de l'arborescence est une entrée dans la variable de type liste ListeFichiers. Nous devons également ajuster la largeur de chaque colonne (à nouveau) pour correspondre à la taille des données de la colonne.

C'est tout pour la première vue de l'application. Exécutez-la et regardez ce que ça fait. Notez que si vous avez un grand nombre de fichiers à parcourir, vous aurez l'impression que le programme ne répond pas. C'est quelque chose qui doit être corrigé. Nous allons créer des routines pour modifier notre curseur de la valeur par défaut à un curseur en forme de montre et vice-versa pour que l'utilisateur soit au courant quand nous faisons quelque chose qui prend beaucoup de temps.

Dans la routine « initialise_var_Tk », ajoutez le code suivant à la fin :

```
global
CurseurOccupe,PreCurseurOccupe,
WidgetsOccupes

CurseurOccupe = 'watch'

PreCurseurOccupe = None

WidgetsOccupes = (racine, )
```

Ici, nous mettons en place des variables globales, nous les initialisons, puis nous réglons le(s) widget(s) (dans WidgetsOccupes) pour lesquels nous souhaitons gérer le changement de curseur. Dans ce cas, nous avons mis racine qui est notre fenêtre entière. Remarquez que c'est un tuple.

```
def ConstruireExts():
    if VchkAVI.get() == '1':
        exts.append(".avi")
    if VchkMKV.get() == '1':
        exts.append(".mkv")
    if VchkMP3.get() == '1':
        exts.append(".mp3")
    if VchkMV4.get() == '1':
        exts.append(".mv4")
    if VchkOGG.get() == '1':
        exts.append(".ogg")
```

Ensuite, nous créons deux routines pour

```
def Parcourir(chemin,extensions):
    rcntr = 0
    liste = []
    for racine, reps, fics in os.walk(chemin):
        rcntr += 1  # nombre de repertoires parcourus
        for fic in [f for f in fics if f.endswith(extensions)]:
            liste.append(fic)
            liste.append(racine)
            ListeFichiers.append(liste)
            liste=[]
```

```
def InitialiserVueArborescente():
    global TitresColonnes
    TitresColonnes = ['Nom fichier','Chemin']
    VueArborescente.configure(columns=TitresColonnes,
        show="headings")
    for col in TitresColonnes:
        VueArborescente.heading(col, text = col.title(),
                command = lambda c = col: sortby(VueArborescente, c, 0))
        ## ajuste la largeur de colonne au titre
        VueArborescente.column(col, width =
tkFont.Font().measure(col.title()))
```

modifier et remettre le curseur. D'abord la routine qui modifie, que nous appelons « DebutOccupation ». Insérez le code que vous voyez au milieu à droite après la routine « ChargerDonnees ».

Nous vérifions d'abord si une valeur a été passée à « nouveaucurseur ». Sinon, nous mettons par défaut CurseurOccupe. Puis nous parcourons le tuple WidgetsOccupes et réglons le curseur sur ce que nous voulons.

Maintenant, mettez le code que vous voyez tout à fait en bas.

Dans cette routine, nous réinitialisons simplement le curseur pour les widgets dans notre tuple WidgetsOccupes à notre curseur par défaut.

Enregistrez et exécutez votre programme. Vous devrez voir que le curseur change chaque fois que vous avez une longue liste de fichiers à parcourir.

Cette application ne fait vraiment pas grand chose, mais elle vous a montré comment utiliser Page pour développer très rapidement. Avec l'article d'aujourd'hui, vous pouvez voir qu'une bonne conception de votre interface graphique à l'avance peut rendre le processus de développement facile et relativement indolore.

Le fichier tcl est enregistré sur pastebin : http://pastebin.com/AA1kE4Dy (en anglais) et le code Python est enregistré ici : http://pastebin.com/WYK2SKQj.

À la prochaine fois !

```
def ChargerDonnees():
    global TitresColonnes
    for c in ListeFichiers:
        VueArborescente.insert('','end',values=c)
        # ajuste la largeur de colonne si necessaire pour chaque valeur
        for ix, val in enumerate(c):
            larg_col = tkFont.Font().measure(val)
            if VueArborescente.column(TitresColonnes[ix],width=None)<larg_col:
                VueArborescente.column(TitresColonnes[ix], width=larg_col)
```

```
def debutOccupation(nouveaucurseur=None):
    global PreCurseurOccupe
    if not nouveaucurseur:
        nouveaucurseur = CurseurOccupe
    nouveauPreCurseursOccupes = {}
    for composant in WidgetsOccupes:
        nouveauPreCurseursOccupes[composant] = composant['cursor']
        composant.configure(cursor=nouveaucurseur)
        composant.update_idletasks()
        PreCurseurOccupe = (nouveauPreCurseursOccupes, PreCurseurOccupe)
```

```
def finOccupation():
    global PreCurseurOccupe
    if not PreCurseurOccupe:
        return
    ancienPreCurseursOccupes = PreCurseurOccupe[0]
    PreCurseurOccupe = PreCurseurOccupe[1]
    for composant in WidgetsOccupes:
        try:
            composant.configure(cursor=ancienPreCurseursOccupes[composant])
        except KeyError:
            pass
        composant.update_idletasks()
```