

# CRÉER UN EXÉCUTABLE À L'AIDE DE Py2exe

## Avec Système d'Exploitation Windows (xp)

Préliminaire : Pour pouvoir réaliser ce tutoriel, il vous faut installer sur votre PC

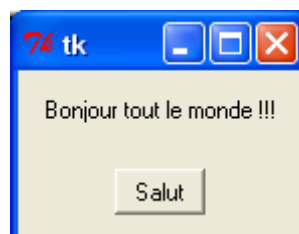
- + Python ; Ça semble évident. Néanmoins, je vous recommande de ne pas utiliser pour l'instant la version 2.5, car actuellement (c'est-à-dire le 10.01.07) il n'existe aucune version de Py2exe pour cette version de Python. Téléchargeable à l'adresse suivante : <http://www.python.org/download/>
- + Pmw ; Cela correspond à Python Mega Widgets. C'est une expansion de la bibliothèque graphique Tkinter. L'installation est un peu compliquée (cf annexe). Vous pouvez télécharger Pmw à l'adresse suivante : <http://prdownloads.sourceforge.net/pmw/Pmw.1.2.tar.gz?download>
- + Py2exe ; C'est un module qui permet de compiler (c'est-à-dire créer un exécutable ; Ici à partir d'un fichier \*.py ou \*.pyw). Vous pouvez le télécharger à l'adresse suivante : <http://www.py2exe.org/old/> ; Allez en bas de la page dans la rubrique “ *Installing py2exe* ”. **Choisissez la version correspondant à celle de python (soit 2.3 soit 2.4)**

## II] Compilation simple

Une fois que vous avez tout installé, nous allons pouvoir passer aux choses sérieuses. Pour commencer, nous allons créer un petit programme utilisant la bibliothèque graphique Tkinter. Crée un fichier “ Exemple\_1.py ” contenant les lignes de code suivantes :

```
1  from Tkinter import *
2
3  fen = Tk()
4
5  texte = Label(fen, text = "Bonjour tout le monde !!! ")
6  texte.pack(padx = 10, pady = 10)
7
8  bouton = Button(fen, text = " Salut ", command = fen.destroy)
9  bouton.pack(padx = 10, pady = 10)
10
11 fen.mainloop()
```

On obtient ce genre de fenêtre :



Pour que nous puissions réaliser notre exécutable, nous allons décider de tout enregistrer dans le répertoire d'installation de Python. Préférez le répertoire de base (C:\Python24). En effet, il ce peut que vous rencontriez quelques *problèmes de détection de répertoire* par Py2exe entre autre du aux accents. Copier donc “ Exemple\_1.py ” dans C:\Python24.

Ensuite, nous allons créer un fichier Python que nous allons appeler “Setup.py”. C’est ce fichier qui va servir à Py2exe à compiler notre fichier “Exemple\_1.py”. Suivant ce que va contenir “Setup.py”, la compilation sera différente.

Dans le fichier “Setup.py” veuillez entrer les lignes de code suivante :

```
1  from distutils.core import setup
2  import py2exe
3
4  setup(console = ["Exemple_1.py"])
```

N’oubliez pas d’enregistrer votre script dans C:\Python24\

Nous allons maintenant créer un fichier \*.bat. Un fichier \*.bat permet d’effectuer des tâches pré-enregistrées par Windows. À noter que l’on pourrait faire la même chose en lançant la console Windows.

*Pour ouvrir la console, veuillez vous reporter à l’Annexe.*

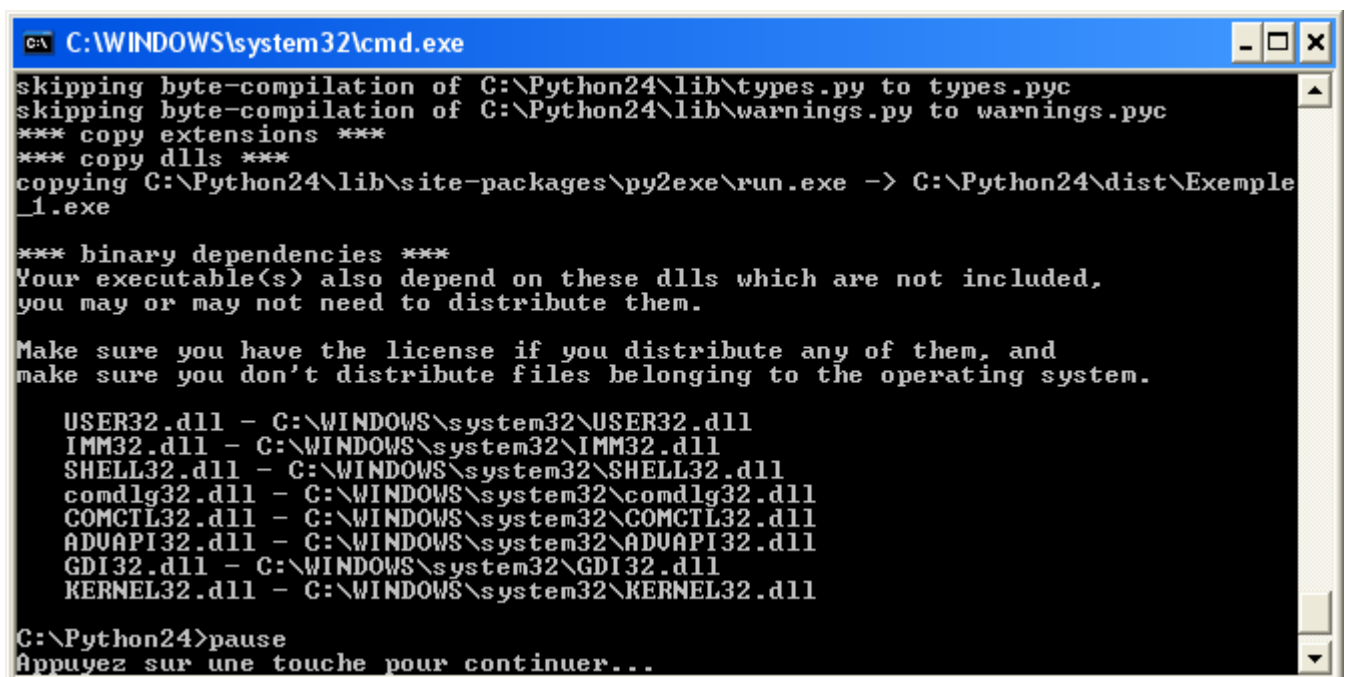
Pour créer un fichier \*.bat, ouvrez un document texte par exemple et tapez les lignes de codes suivantes :

```
"C:\Python24\python.exe" "C:\Python24\Setup.py" py2exe
pause
```

« pause » permet d’arrêter le programme pour visualiser les erreurs s’il y en a.

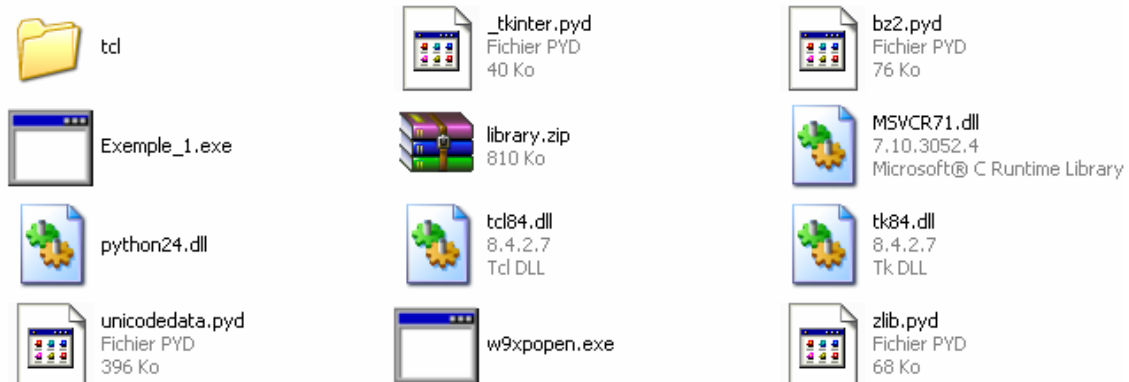
Enregistrez votre fichier “Exécution.bat” (si vous l’avez tapé dans un document \*.txt n’oubliez pas de changer l’extension) dans C:\Python24\.

Maintenant lancez “Exécution.bat”. Si vous n’avez pas fait d’erreur, voici ce qui se passe : La console se lance et plusieurs opérations s’effectuent.

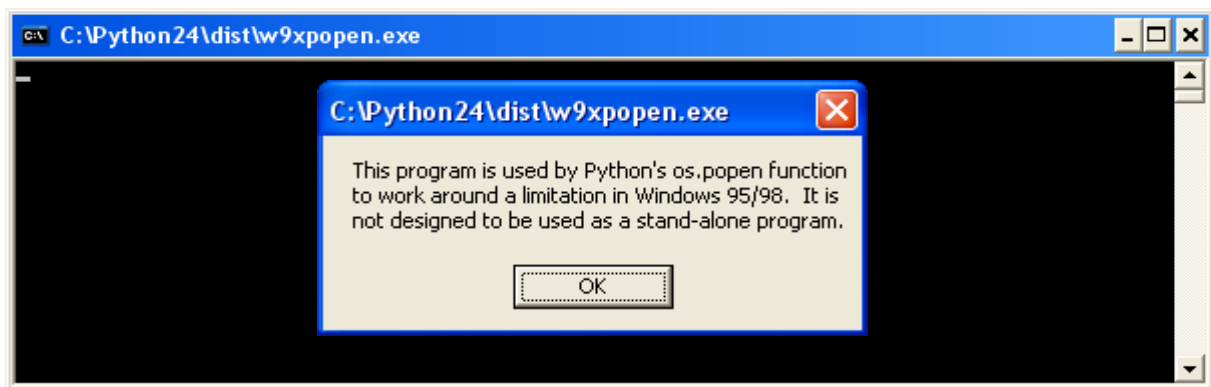


```
C:\WINDOWS\system32\cmd.exe
skipping byte-compilation of C:\Python24\lib\types.py to types.pyc
skipping byte-compilation of C:\Python24\lib\warnings.py to warnings.pyc
*** copy extensions ***
*** copy dlls ***
copying C:\Python24\lib\site-packages\py2exe\run.exe -> C:\Python24\dist\Exemple_1.exe
*** binary dependencies ***
Your executable(s) also depend on these dlls which are not included,
you may or may not need to distribute them.
Make sure you have the license if you distribute any of them, and
make sure you don't distribute files belonging to the operating system.
USER32.dll - C:\WINDOWS\system32\USER32.dll
IMM32.dll - C:\WINDOWS\system32\IMM32.dll
SHELL32.dll - C:\WINDOWS\system32\SHELL32.dll
comdlg32.dll - C:\WINDOWS\system32\comdlg32.dll
COMCTL32.dll - C:\WINDOWS\system32\COMCTL32.dll
ADVAPI32.dll - C:\WINDOWS\system32\ADVAPI32.dll
GDI32.dll - C:\WINDOWS\system32\GDI32.dll
KERNEL32.dll - C:\WINDOWS\system32\KERNEL32.dll
C:\Python24>pause
Appuyez sur une touche pour continuer...
```

Un dossier nommé “ dist” se crée dans le même répertoire que notre fichier “Setup.py ” à savoir C:\Python24\ . Il contient alors ceci



L’exécutable “ Exemple\_1.exe ” est celui que nous souhaitons créer. Vous vous apercevrez qu’un autre exécutable est créé : “ w9xpopen.exe ”. Si vous essayez de le lancer voici ce qui se passe :

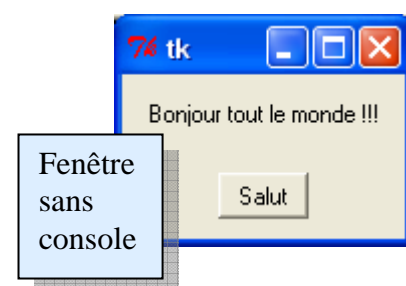


Cette fenêtre s’affiche. Il semblerait que ce programme soit indispensable sous Windows 95/98, mais je n’en suis pas sûr. Néanmoins, si vous le supprimez ou si vous ne le mettez pas dans le même dossier que “ Exemple\_1.exe ”, ce dernier ne fonctionne plus. De même pour tous les autres fichiers créés.

Remarque : la console se lance à l’exécution :

Si vous ne souhaitez pas qu’elle s’affiche il faut modifier votre fichier “Setup.py ” (remplacer *console* par *windows* la ligne n°4) :

```
4  setup(windows = ["Exemple_1.py"])
```



Remarque : Si votre programme ne contient aucune interface graphique, laisser la console sinon, vos utilisateurs ne pourront plus communiquer avec votre programme

Une icône est une petite image qui représente un programme. Elle est présente en haut à gauche de la fenêtre du programme :

Également dans la barre de tâche :



Il représente les raccourcis, des fichiers et exécutables : ici un fichier Python  
Ici l'icône classique d'un exécutable



Exemple\_1.exe



Exemple\_1.py

Notre but est que dans la fenêtre, la barre de tâche et en représentation de l'exécutable de notre programme “ Exemple\_1.py ”, l'icône de Python s'affiche.

Un icône, se présente sous la forme d'un fichier image d'expansion \*.ico ; Celui de Python se nomme “py.ico” est il présent dans le répertoire d'installation.

Modifions notre fichier “ Exemple\_1.py ” que nous appellerons maintenant “ Exemple\_2.py ”. Nous allons modifier l'icône de la fenêtre et de la barre de tâche à l'aide de cette ligne : `fen.iconbitmap("py.ico")` (voir ligne n°4)

```
1  from Tkinter import *
2
3  fen = Tk()
4  fen.iconbitmap("py.ico")
5
6  texte = Label(fen, text = "Bonjour tout le monde !!! ")
7  texte.pack(padx = 10, pady = 10)
8
9  bouton = Button(fen, text = " Salut ", command = fen.destroy)
10 bouton.pack(padx = 10, pady = 10)
11
12 fen.mainloop()
```

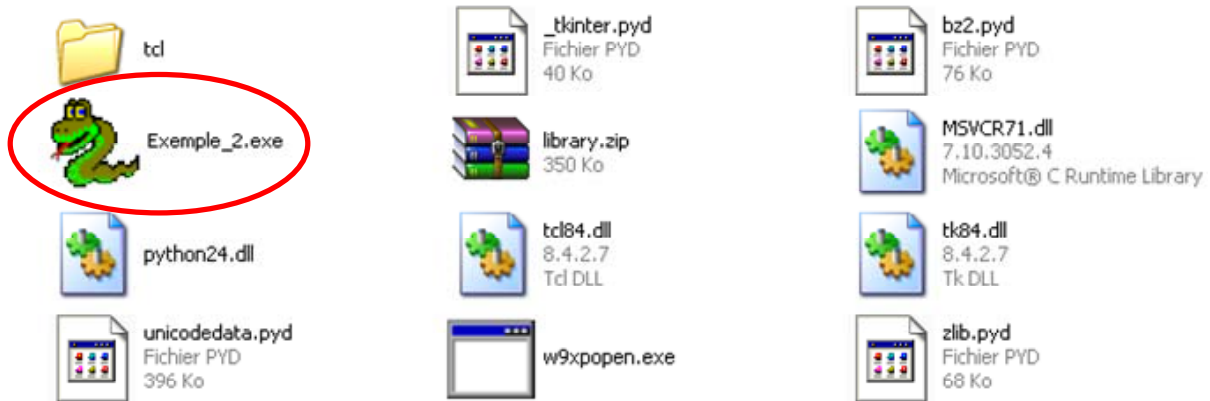
Il faut bien entendu que “py.ico” soit dans le même répertoire que “ Exemple\_2.py ”.

Modifions maintenant notre fichier “Setup.py”

```
1  from distutils.core import setup
2  import py2exe
3
4
5  setup(windows=[{"script": "Exemple_2.py", "icon_resources": [(1, "py.ico")]}],
6        options = {"py2exe":
7                    {"compressed": 2,
8                     "optimize": 1}})
```

Lancer la compilation en double cliquant sur “ Exécution.bat ”. N’oublier pas de supprimer au préalable votre dossier “ dist” qui ne nous sert plus à rien. Si vous ne le supprimer pas, il se peut que vous ayez les anciens exécutable en plus des nouveaux. Vérifier également que l’icône “py.ico ” reste dans C:\Python24\.

Après compilation allez dans le dossier “ dist” et là vous remarquerez quelque chose : notre exécutable est symboliser par l’icône Python ! Nous avons réussi !



Évidement, vous lancez notre exécutable est la .....  
Ç A N E F O N C T I O N N E P A S ! ! !

Pourquoi ??? Et bien, tout simplement parce que votre fichier “py.ico ” n’est pas dans le même répertoire que votre exécutable. Copier le dans *dist* et essayer à nouveau. Cette fois ci tout va bien !

## 2/ Options supplémentaires

Le module Py2exe offre une multitude d’options, dont je ne connais certainement même pas  $\frac{1}{8}$ . Nous allons voir comment mettre un commentaire à notre exécution. Pour cela, nous devons modifier notre fichier “Setup.py ” :

```
1  import glob
2  import os
3  import re
4  from distutils.core import setup
5  import py2exe
6
7
8  setup(windows=[{"script": "Exemple_2.py",
9                  "other_resources": [(u"VERSIONTAG", 1, "1.0")],
10                 "icon_resources": [(1, "py.ico")]}],
11
12        description = "Logiciel test (par Aéra group).",
13
14        version = "1.0",
15
16        options = {"py2exe":
17                   { "compressed": 1,
18                     "optimize": 2,
19                     "excludes": ["_gtkagg", "_tkagg"],
20                     "dll_excludes": ["libgdk-win32-2.0-0.dll",
21                                       "libgobject-2.0-0.dll"]}}})
```

Et voila ce qu’on obtient :



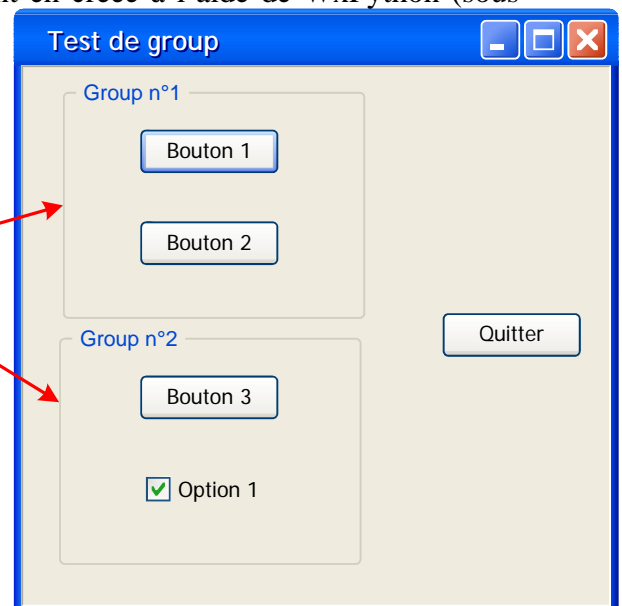
### III] Remarques

#### 1/ Détection des bibliothèques

Créons un fichier nommé “ Exemple\_3.py ” dans lequel nous utiliserons la bibliothèque Pmw. Nous allons créer une fenêtre avec un label et un bouton comme dans le premier exemple, mais en plus nous allons ajouter un group : c’est un widget<sup>1</sup> qui permet de ranger les éléments dans des « boîtes ». Vous pouvez également en créer à l’aide de WxPython (sous Pmw ils sont un peu différents)



Group (style xp) créé avec WxPython



```

1 from Tkinter import *
2 import Pmw
3
4 fen = Tk()
5
6 g=Pmw.Group(fen, tag_text="Titre du group")
7 g.pack(padx = 10, pady = 10)
8
9 texte = Label(g.interior(), text = "Bonjour tout le monde !!!")
10 texte.pack(padx = 10, pady = 10)
11
12 bouton = Button(g.interior(), text = " Salut ", command = fen.destroy)
13 bouton.pack(padx = 10, pady = 10)
14
15 fen.mainloop()

```

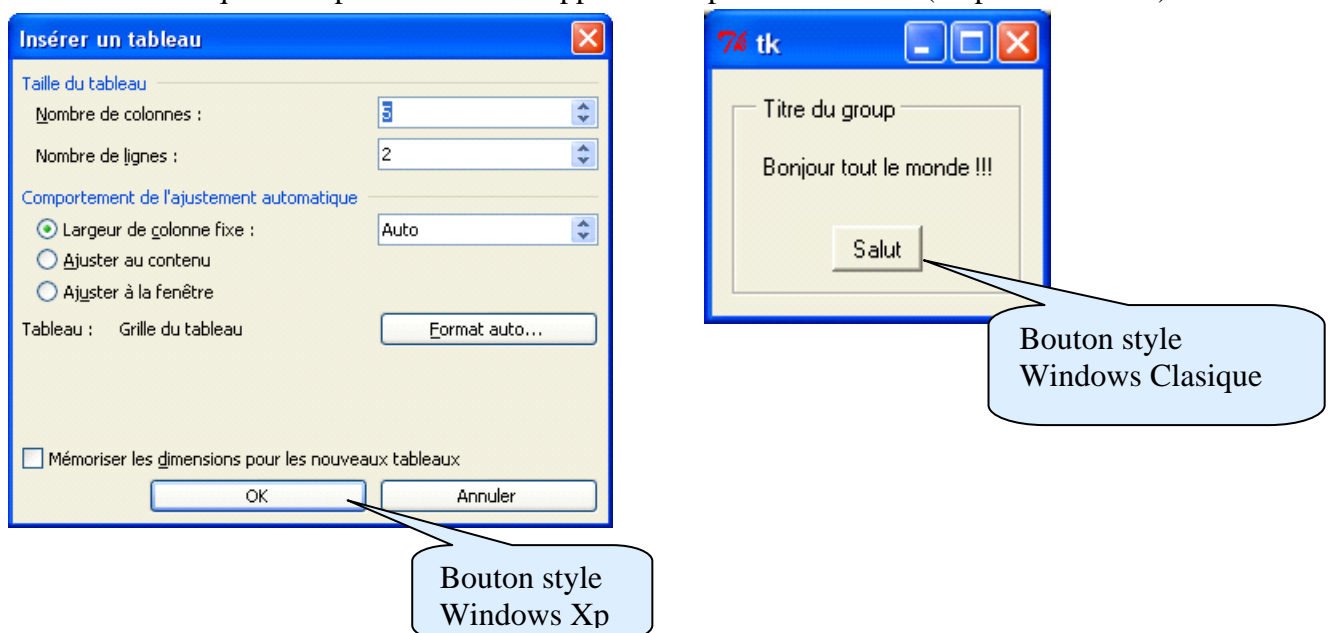
1. **Widget** : En programmation objet graphique tel que des boutons.

Créons maintenant un exécutable de se programme avec la première méthode (pour plus de simplicité). Et là, vous remarquerez que notre exécutable ne fonctionne pas !

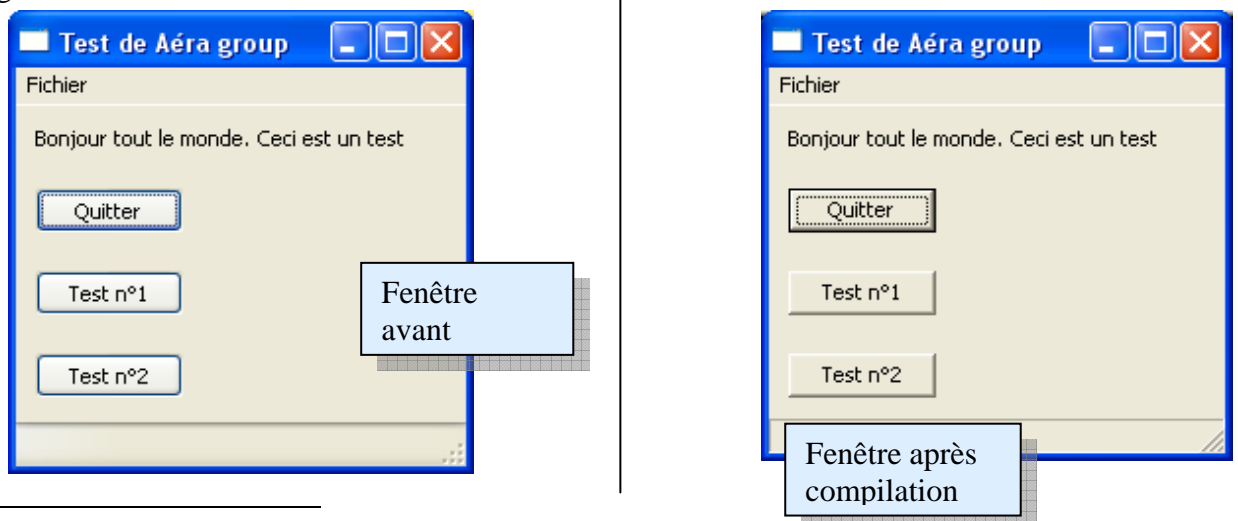
En fait, c'est la bibliothèque Pmw qui pose problème : Py2exe n'arrive pas à la détecter ni à l'inclure dans les fichiers nécessaire à l'exécution de notre programme. Il y a évidemment pas que la bibliothèque Pmw qui soit consterné ! La plus part des modules qui ne sont pas présent de base dans Python ne sont pas débecter (sauf WxPython). Nous allons voir plus long comment résoudre se problème dans la plus par des cas.

## 2/ Avec WxPython<sup>2</sup>

J'ai créé un petit programme avec WxPython. Vous avez remarqué que pour l'instant que les programmes que nous avons compilés étaient identiques à ceux non compilés (graphiquement parlant). Et bien, avec WxPython, c'est un peu différent. Sous Windows Xp, vous avez sans doute remarqué que les programmes doter d'une interface graphique Tkinter, n'avez pas les mêmes boutons que ceux présent dans les applications professionnelles (ici pris de Word<sup>®</sup>) :



Avec WxPython, vous avez sans doute remarqué, que les widgets sont du style Xp. Comparer la fenêtre avant compilation et après compilation, vous allez vous apercevoir que le style à changer.



2. Vous pouvez télécharger **WxPython** à cette adresse : <http://www.wxpython.org/download.php>

Pour résoudre ce problème, une solution existe : dans le répertoire d'installation de Python (C:\Python24 le plus souvent) il existe un fichier nommé **python.exe.manifest** (à ne pas confondre avec **pythonw.exe.manifest** qui se trouve dans le même répertoire). Imaginons que votre application soit nommée “ Exemple\_4.py ”



```
1  import wx
2
3  class MyFrame(wx.Frame):
4      def __init__(self, parent, title):
5          wx.Frame.__init__(self, parent, -1, title)
6
7
8
9          menuBar = wx.MenuBar()
10
11
12          menu = wx.Menu()
13
14
15
16          menuBar.Append(menu, "Fichier")
17          self.SetMenuBar(menuBar)
18
19          self.CreateStatusBar()
20
21
22
23          panel = wx.Panel(self)
24
25          t = wx.StaticText(panel, -1, "Bonjour tout le monde. Ceci est un test")
26          t.SetSize(t.GetBestSize())
27          b1 = wx.Button(panel, -1, "Quitter")
28          b2 = wx.Button(panel, -1, "Test n°1")
29          b3 = wx.Button(panel, -1, "Test n°2")
30
31
32          sizer = wx.BoxSizer(wx.VERTICAL)
33          sizer.Add(t, 0, wx.ALL, 10)
34          sizer.Add(b1, 0, wx.ALL, 10)
35          sizer.Add(b2, 0, wx.ALL, 10)
36          sizer.Add(b3, 0, wx.ALL, 10)
37          panel.SetSizer(sizer)
38          panel.Layout()
39
40          self.Centre()
41
42
43
44  class MyApp(wx.App):
45      def OnInit(self):
46          frame = MyFrame(None, "Test de Aéra group")
47          self.SetTopWindow(frame)
48
49          frame.Show(True)
50          return True
51
52
53  app = MyApp(True)
54  app.MainLoop()
```



Compiler à l'aide de la méthode simple et copier **python.exe.manifest** dans *dist*. Renommer ce dernier en **Exemple\_4.exe.manifest**. Maintenant, si vous lancer “ Exemple\_4.exe ”, vous obtiendrez une interface graphique comme vous avez l'avoir à l'aide de WxPython dans le script “ Exemple\_4.py ”. Dans le cas général, vous devrez renommer le fichier manifest en **le\_nom\_de\_votre\_programme.exe.manifest**.

### 3/ Importer des modules complémentaires

Vous vous êtes rendu compte précédemment que certaines bibliothèques ne sont pas correctement détecter ! Ce n'est pas le cas des bibliothèques présent de base avec Python, ni de WxPython. Mais par exemple ; une application créé avec Pmw, n'est pas correctement détecter. Il faut forcer Py2exe à détecter ces bibliothèques. Je ne sais pour l'instant que la méthode, mais je ne l'ai jamais testé. De plus, Pmw à un système d'importation très particulier et donc, la technique que je vais vous monterez ne fonctionnera pas pour les applications créé à l'aide de Pmw. Si quelqu'un à une information, merci d'en faire profiter les autres en laissant un commentaire !

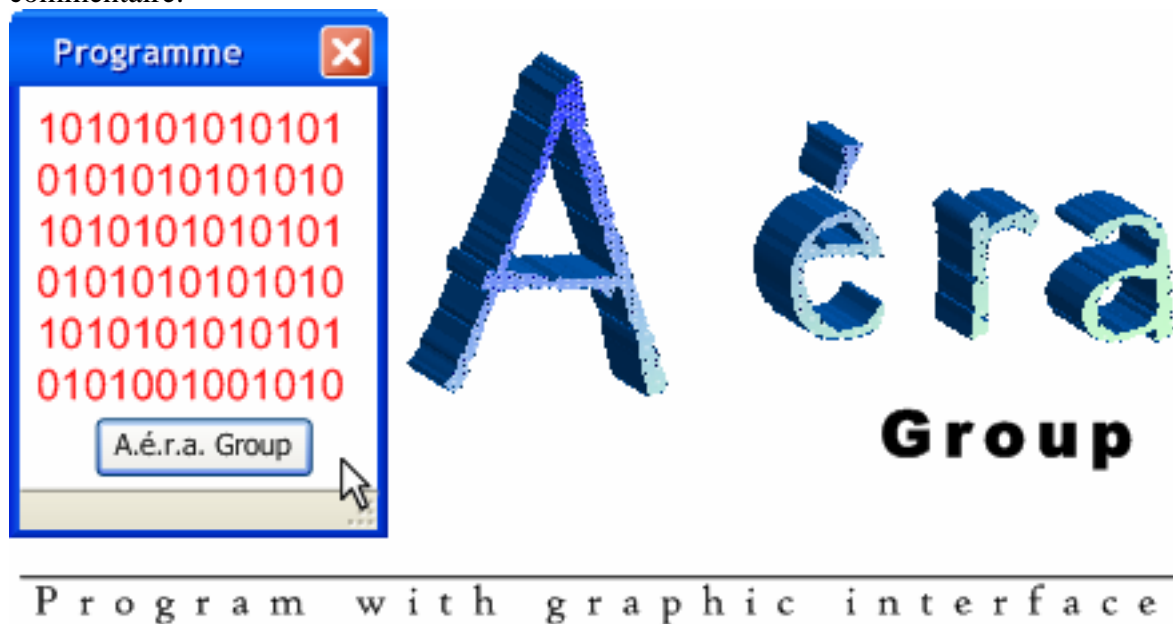
Il nous faut modifier notre fichier “Setup.py ”. Pour cette exemple, imaginer que la librairie à importer est par exemple la librairie WxPython (même si on sait que ce n'est pas utile)

```
1  import glob
2  import os
3  import re
4  from distutils.core import setup
5  import py2exe
6
7
8  setup(windows=[{"script": "Exemple_4.py",
9                  "other_resources": [(u"VERSIONTAG",1, "1.0")]}],
10
11        description = "Logiciel test (par Aéra group).",
12
13        version = "1.0",
14
15        options = {"py2exe":
16                    { "compressed": 1,
17                      "optimize": 2,
18                      "excludes": ["wxPython.wx"],
19                      "dll_excludes": ["libgdk-win32-2.0-0.dll", "libgobject-
20                                     2.0-0.dll"],
21                      "packages": ["wx"]
```

Les lignes en gras sont les plus importante. C'est celle-ci que vous devez modifier pour que le fonctionnement soit correct. Pour savoir quelles informations doivent être apportées à ces lignes, veuillez effectuer une recherche sur le Web.

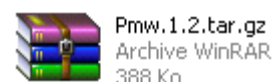
Tout au long ce tutoriel, vous avez plus vous rendre compte que la compilation est un peu compliquer sous Python. J'ai essayé de faire ce tutoriel le plus simplement possible, J'espère que vous avez compris l'essentiel. Je sais également que Py2exe n'est pas la seule solution pour compiler, je connais de nom **PyInstallleur**. Je ne l'ai j'aimais utiliser - <http://pyinstaller.hpcf.upr.edu/cgi-bin/trac.cgi>

Ne faite pas attention aux fautes d'orthographes ; Si vous rencontrer un problème laisser un commentaire.



## A N N E X E

### 1/ Installation de Pmw

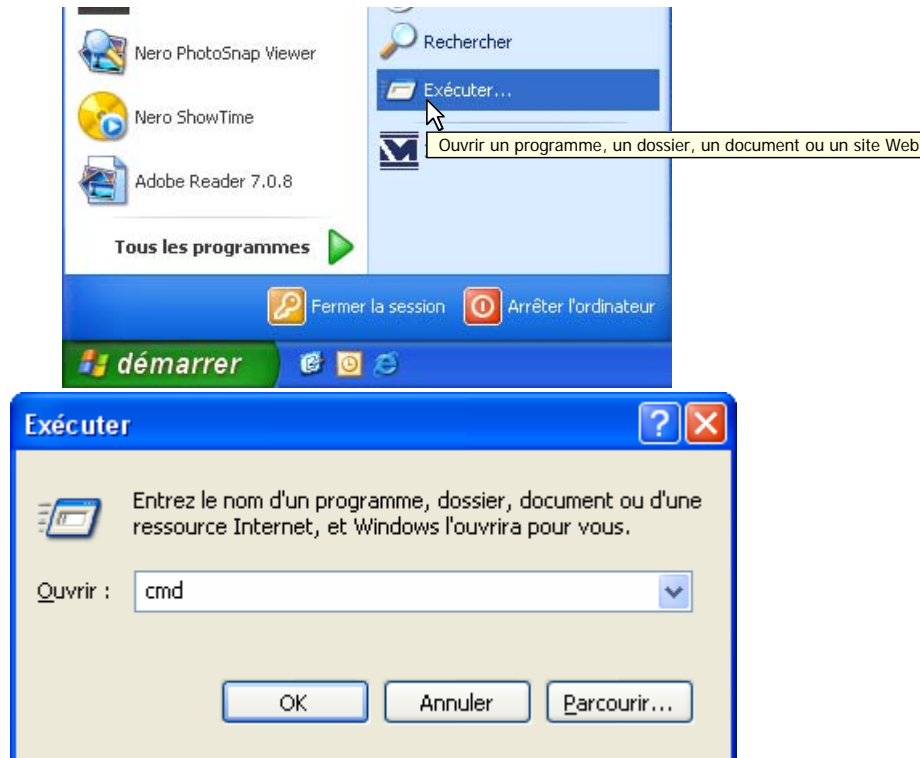


Une fois que vous avez téléchargé le ZIP du site indiquer préliminairement, copier celui-ci dans *C:\Python24\Lib\site-packages* si *C:\Python24\* est votre répertoire d'installation de Python. Décompressez l'archive dans ce répertoire, et théoriquement, vous avez réussi l'installation de Pmw.

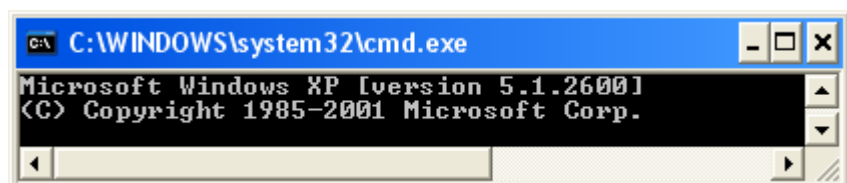
## 2/ Lancement de la console sous Windows Xp

Allez dans le menu démarrer de Windows et sélectionner Exécuter.

Une boîte de dialogue s'affiche. Taper dans le champ d'entrée, **cmd** et cliquer sur OK.



La console est lancée.



By Aéra Group