

Construire une interface graphique pas à pas en Python avec Tkinter et wxPython

http://sebsauvage.net/python/gui/index_fr.html

Contenu

Table des matières	3
Notre projet.....	3
Etape 1 : Importer le toolkit	4
Etape 2 : Créer une classe	5
Etape 3 : Le constructeur.....	5
Etape 4 : Garder une référence de notre parent	6
Etape 5 : Initialiser l'interface graphique	7
Etape 6 : Création du main.....	8
Etape 7 : Boucler !	9
Etape 8 : Gestionnaire de layout	11
Etape 9 : Ajouter un champ texte	12
Etape 10 : Ajouter le bouton	15
Etape 11 : Ajouter le label	17
Etape 12 : Activer le redimensionnement automatique.....	19
Etape 13 : Ajouter une contrainte.....	21
Etape 14 : Ajouter des gestionnaires d'évènements.....	23
Etape 15 : Modifier le label	27
Etape 16 : Afficher la valeur	29
Etape 17 : Petit raffinement.....	32
Etape 18 : Correction d'un comportement de Tkinter	34
C'est terminé !.....	37
Outils RAD et coordonnées pixel.....	37
Interface graphique non bloquante ?	38

Tkinter ou wxPython ?.....	39
Autres toolkit graphiques	40
Créer un fichier EXE	40
A propos de ce document	40

Ce texte est la traduction française de <http://sebsauvage.net/python/gui/index.html>.

Dans cette page, vous allez apprendre à construire une interface graphique **pas à pas** en Python.

Le but est:

- De maîtriser les techniques de base des interfaces graphiques (mise en place des widgets, contraintes, gestion des événements...)
- **De comprendre la *moindre* méthode et paramètre utilisé ici.**
- De voir deux toolkits majeurs et de comprendre leurs différences.
- De servir de base pour construire vos propres applications graphiques.

Vous apprendrez:

- à créer une classe "application graphique",
- à créer des widgets (éléments d'interface graphiques),
- à les placer dans des conteneurs,
- à attacher des méthodes à certains événements,
- à manipuler les valeurs des widgets,
- etc.

Nos contraintes pour ce document sont:

- Faire la même chose avec **Tkinter** (le toolkit graphique fourni en standard avec Python) et **wxPython** (un toolkit portable, populaire et plus avancé).
- Programmer le tout de la *bonne* manière.
- Utiliser un français clair.
- Explicite est mieux qu'implicite ;-)

Tkinter englobe la librairie **Tcl/tk** afin que nous puissions l'utiliser en Python. De la même manière, **wxPython** englobe la librairie **wxWidgets** afin qu'on puisse l'utiliser en Python.

Nous utiliserons un terme ou l'autre sans distinction dans ce document.

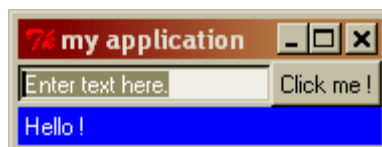
Ces deux toolkits graphiques sont portables: Cela signifie que - si bien conçues - vos interfaces fonctionneront sur tous les systèmes (Windows, Linux, MacOS X...).

Table des matières

- [Notre projet](#)
- [Etape 1 : Importer le toolkit](#)
- [Etape 2 : Créer une classe](#)
- [Etape 3 : Le constructeur](#)
- [Etape 4 : Garder une référence de notre parent](#)
- [Etape 5 : Initialiser l'interface graphique](#)
- [Etape 6 : Création du main](#)
- [Etape 7 : Boucler !](#)
- [Etape 8 : Gestionnaire de layout](#)
- [Etape 9 : Ajouter un champ texte](#)
- [Etape 10 : Ajouter le bouton](#)
- [Etape 11 : Ajouter le label](#)
- [Etape 12 : Activer le redimensionnement automatique](#)
- [Etape 13 : Ajouter une contrainte](#)
- [Etape 14 : Ajouter des gestionnaires d'évènement](#)
- [Etape 15 : Modifier le label](#)
- [Etape 16 : Afficher la valeur](#)
- [Etape 17 : Petit raffinement](#)
- [STEP 18 : Correction d'un comportement de Tkinter](#)
- [C'est terminé !](#)
- [Outils RAD et coordonnées pixel](#)
- [Interface graphique non bloquante ?](#)
- [Tkinter ou wxPython ?](#)
- [Autres toolkit graphiques](#)
- [Créer un fichier EXE](#)
- [A propos de ce document](#)

Notre projet

Notre projet est de créer une application toute simple comme celle-là:



Un **widget** est un élément graphique (un bouton, une case à cocher, un label, une zone de texte, un onglet...)

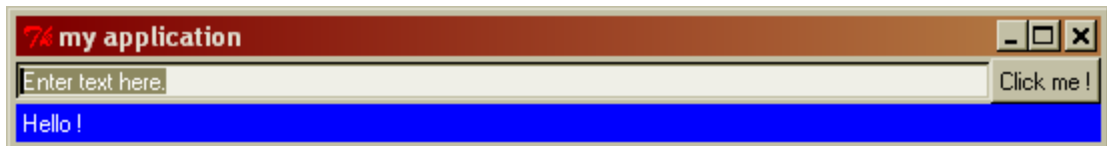
Notre application possède 3 widgets:

- Un champ texte ("Enter text here")
- Un bouton ("Click me !")

- Un label bleu ("Hello !")

Le comportement que nous souhaitons avoir est:

- Quand la touche ENTREE est pressée dans le champ texte, ou que le bouton est cliqué, le label bleu affiche le texte qui a été entré.
- La fenêtre ne doit être redimensionnable que horizontalement.
- Si la fenêtre est redimensionnée, le champ texte et le label bleu doivent s'agrandir en conséquence.



Dans ce document, nous montrerons côte à côte le code Tkinter et wxPython de cette manière:

Le code **Tkinter** est dans cette couleur, à **gauche**.

Le code **wxPython** équivalent est dans cette couleur, à **droite**.

Les lignes ajoutées à chaque étape seront mise en valeur **de cette manière**.

Bien, allons-y !

Etape 1 : Importer le toolkit

Importons le module dont nous avons besoin:

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

try:

```
import wx
```

except ImportError:

```
raise ImportError,"The wxPython module is required to run this program."
```

Tkinter fait partie de la distribution standard de Python. On s'attend donc à ce qu'il soit présent. Nous nous contentons de l'importer.

wxPython ne fait pas partie de la distribution standard de Python et doit être téléchargé et installé séparément. Il est plus convenable de prévenir l'utilisateur que notre programme nécessite wxPython mais qu'il n'est pas installé, et où le trouver (d'où le bloc try/except ImportError).

Etape 2 : Créer une classe

C'est mieux de mettre notre application dans une classe:

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    #!/usr/bin/python
    # -*- coding: iso-8859-1 -*-

    try:
        import wx
    except ImportError:
        raise ImportError, "The wxPython module is required to run this program"

    class simpleapp_wx(wx.Frame):
```

En **Tkinter**, on hérite de **Tkinter.Tk** qui est la classe de base pour les fenêtres standards.

En **wxPython**, on dérive de **wx.Frame** qui est la classe de base pour les fenêtres standards.

Etape 3 : Le constructeur

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
```

```

def __init__(self,parent):
    Tkinter.Tk.__init__(self,parent)

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)

```

simpleappTk dérive de **Tkinter.Tk**, nous devons donc appeler le constructeur de **Tkinter.Tk** (**Tkinter.Tk.__init__()**).

simpleapp_wx dérive de **wx.Frame**, nous devons donc appeler le constructeur de **wx.Frame** (**wx.Frame.__init__()**).

Une interface graphique est une hiérarchie d'objets: Un bouton peut être contenu dans un panneau qui est contenu dans un onglet qui est contenu dans une fenêtre, etc.

Donc chaque élément de l'interface graphique (widget) possède un parent (le widget qui le contient, généralement).

C'est pour cela que chacun de nos constructeurs a un paramètre **parent**.

Garder la référence du parent est utile quand il faut montrer/masquer des groupes de widgets, les redessiner à l'écran ou tout simplement les détruire quand la fenêtre est fermée.

L'objet **wx.Frame** a deux paramètres supplémentaires: **id** (un identifiant du widget, que nous n'utiliserons pas) et **title** (le titre de la fenêtre).

Etape 4 : Garder une référence de notre parent

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleappTk(Tkinter.Tk):
    def __init__(self,parent):

```

```

        Tkinter.Tk.__init__(self,parent)
        self.parent = parent

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent

```

C'est une bonne habitude à prendre de garder une référence de notre parent quand on crée un widget.

Etape 5 : Initialiser l'interface graphique

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

```

```

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.Show(True)

```

Il est généralement préférable de garder la portion de code qui **créé les éléments graphique** (boutons, champs texte...) séparée de la **logique** du programme.
C'est pour cela que nous créons une méthode **initialize()**. Nous allons créer tous nos widgets dans cette méthode.

Pour le moment, la version **Tkinter** ne contient rien (d'où l'instruction **pass** qui ne fait rien.)
La version **wxPython** contient **self.Show(True)** pour obliger la fenêtre à apparaître (sans quoi elle reste cachée après sa création). (Ceci n'est pas nécessaire avec Tkinter car Tkinter affiche automatiquement tous les widgets créés).

Etape 6 : Création du main

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')

```



```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError, "The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None, -1, 'my application')
```

Maintenant que nous avons notre classe, utilisons-la !

Nous créons un **main** qui sera exécuté quand le programme sera lancé à partir de la ligne de commande.

Avec **Tkinter**, nousinstancions notre classe (**app=simpleapp_tk()**). On ne lui donne aucun parent (**None**), car c'est le premier élément graphique que nous créons. On donne également un titre à notre fenêtre (**app.title()**).

Avec **wxPython**, est il obligatoire d'instancier un objet **wx.App()** avant de créer des éléments graphiques. C'est pour cela que nous faisons **app=wx.App()**.

Ensuite nousinstancions notre classe (**frame=simpleapp_wx()**). Nous ne lui donnons également aucun parent (**None**), car c'est le premier élément graphique que nous créons.

On utilise **-1** pour laisser wxPython choisir un identifiant lui-même. Et nous donnons à notre fenêtre son titre: **'my application'**.

Etape 7 : Boucler !

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Maintenant, nous demandons à nos deux programme de boucler avec **.mainloop()**

Qu'est-ce que ça signifie ?

Cela veut dire que chaque programme va maintenant **boucler sans fin**, en attente d'évènements (l'utilisateur qui clique sur un bouton, presse une touche, le système d'exploitation qui demande à

notre application de quitter, etc.)

Les boucles Tkinter et wxPython recevront ces évènements et agiront en conséquence.

C'est de la **programmation évènementielle** (Car chaque programme ne fait rien d'autre qu'attendre des évènements, et ne réagit que quand il en reçoit un.)

✿ ***A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !***

Ils afficheront une fenêtre vide.

Maintenant, fermez ces fenêtres et revenons au code source pour y ajouter nos widgets.

Etape 8 : Gestionnaire de layout

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()
```

```
    def initialize(self):
        self.grid()
```

```
if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
```

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"
```

```
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
```

```

wx.Frame.__init__(self,parent,id,title)
self.parent = parent
self.initialize()

def initialize(self):
    sizer = wx.GridBagSizer()
    self.SetSizerAndFit(sizer)
    self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Il y a plusieurs manières de mettre les widgets dans une fenêtre (ou un autre conteneur): Les ajouter horizontalement, les empiler verticalement, etc.

Il existe donc différentes classes (appelées **gestionnaires de layout**, ou ***layout managers***) capables de placer les widgets dans les conteneurs de différentes manières. Certains sont plus souples que d'autres.

Chaque conteneur (fenêtre, panneau, onglet, dialogue...) peut avoir son propre gestionnaire de layout.

Je recommande le gestionnaire **grid** (grille). C'est une simple grille où vous positionnez vos widgets dans des cases à la manière d'un tableur (Excel, OpenOffice Calc...)

Par exemple: Mettre le bouton à la colonne 2, ligne 1. Mettre une checkbox colonne 5, ligne 3. etc.

Si vous ne devez en apprendre qu'un, prenez celui-là.

Avec **Tkinter**, l'appel à **self.grid()** va automatiquement créer un le gestionnaire de layout grille et demander à notre fenêtre de l'utiliser.

Avec **wxPython**, nous créons explicitement le gestionnaire avec **sizer=wx.GridBagSizer()** et nous demandons à notre fenêtre de l'utiliser (**self.SetSizerAndFit(sizer)**).

Maintenant ajoutons des widgets.

Etape 9 : Ajouter un champ texte

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

```

```

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()

```

```
frame = simpleapp_wx(None,-1,'my application')
app.MainLoop()
```

Pour ajouter un widgets, vous devez toujours:

- d'abord **créer** le widget
- puis l'**ajouter** à un conteneur.

D'abord, créons le widget:

Avec **Tkinter**, on crée le widget **Entry** (**self.entry=Tkinter.Entry()**)

Avec **wxPython**, on crée le widget **TextCtrl** (**self.entry=wx.TextCtrl()**)

Dans les deux cas, on passe **self** comme **parent**, car notre fenêtre sera le parent de ces widgets: Ils apparaîtront dans notre fenêtre.

wxPython.**TextCtrl** a 2 paramètres supplémentaires: **-1** (pour que wxPython choisisse automatiquement un identifiant), et le texte lui-même (**u'Enter text here.'**).

(Pour le texte dans Tkinter.Entry, nous verrons cela plus tard.)

Notez que nous conservons une référence au widget que nous venons de créer (**self.entry=...**) car nous aurons besoin d'y accéder plus tard, dans d'autres méthodes de notre application.

Maintenant il est temps de l'ajouter au conteneur:

Avec **Tkinter**, on appelle la méthode **.grid()** sur notre widget. Nous indiquons où le mettre dans la grille (**column=0, row=0**).

Si la cellule doit s'agrandir, on peut demander aux widgets qu'elle contient de rester collés à un des bords de la cellule. C'est option **sticky='EW'**.

(E=east (gauche), W=West (droite), N=North (haut), S=South (bas))

Nous avons spécifié 'EW', ce qui veut dire que notre widget essaiera de rester collé aux bords gauche et droit de sa cellule.

(C'est l'un de nos buts: Que le champ texte s'agrandisse quand on redimensionne la fenêtre horizontalement.)

Avec **wxPython**, nous appelons la méthode **.Add()** sur le conteneur (la fenêtre). Nous lui passons le widget que nous venons de créer (**self.entry**) et ses coordonnées (**0,0**).

(1,1) est l'étendue de la cellule: Dans notre cas, notre cellule ne déborde pas sur les cellule voisines (d'où le (1,1)).

wx.EXPAND demande au gestionnaire de fenêtres d'agrandir notre champ texte si sa cellule est agrandie.

✿ **A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !**

Nous avons une fenêtre avec un simple champ texte. Vous pouvez déjà taper du texte dedans.

Hé ! Le champ texte ne s'agrandit pas quand je redimensionne la fenêtre ! Vous avez menti !

Restez calme, il y a une excellente raison à cela: Nous avons demandé à nos champs texte de s'agrandir si la **cellule ou colonne** qui les contient change de taille, mais nous n'avons pas encore demandé à notre **gestionnaire de layout** lui-même d'élargir les colonnes (et donc les cellules) si la fenêtre est redimensionnée. Nous verrons cela plus tard.

Etape 10 : Ajouter le bouton

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = Tkinter.Button(self,text=u"Click me !")
        button.grid(column=1,row=0)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
```

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError, "The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self, -1, value=u"Enter text here.")
        sizer.Add(self.entry, (0,0), (1,1), wx.EXPAND)

        button = wx.Button(self, -1, label="Click me !")
        sizer.Add(button, (0,1))

        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None, -1, 'my application')
    app.MainLoop()

```

C'est facile dans les deux cas: Créer le bouton, et l'ajouter.

Notez que dans ce cas, on ne conserve pas de référence au bouton (car nous n'aurons pas besoin de lire ou modifier sa valeur par la suite).

 ***A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !***

Nous avons une fenêtre avec un champ texte et un bouton.

Etape 11 : Ajouter le label

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = Tkinter.Button(self,text=u"Click me !")
        button.grid(column=1,row=0)

        label = Tkinter.Label(self,
                               anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
```

```

def initialize(self):
    sizer = wx.GridBagSizer()

    self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
    sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

    button = wx.Button(self,-1,label="Click me !")
    sizer.Add(button, (0,1))

    self.label = wx.StaticText(self,-1,label=u'Hello !')
    self.label.SetBackgroundColour(wx.BLUE)
    self.label.SetForegroundColour(wx.WHITE)
    sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

    self.SetSizerAndFit(sizer)
    self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Nous créons également un label:

- C'est un objet **Label** en **Tkinter**.
- C'est un objet **StaticText** en **wxPython**.

Pour la couleur: On prend un texte blanc sur fond bleu.

- Avec **Tkinter**, c'est **fg="white",bg="blue"**.
- Avec **wxPython**, il faut appeler deux méthodes (**SetForegroundColour** and **SetBackgroundColour**)

Pour l'alignement du texte:

- Avec **Tkinter**, **anchor="w"** signifie que le texte doit être aligné à gauche (w=west=ouest) dans le label.
- Avec **wxPython**, le texte est aligné par défaut à gauche. Nous n'avons donc rien à faire.

Concernant le placement du label dans la grille:

- Avec **Tkinter**, c'est à nouveau la méthode **.grid()**, mais cette fois la cellule doit déborder sur sa voisine de droite (afin que le label apparaisse sous le champ texte (colonne 0) **et** le bouton (colonne 1).): C'est le paramètre **columnspan=2**.
- Nous faisons la même chose avec **wxPython** en spécifiant **(1,2)** (ce qui veut dire: *s'étirer d'une cellule verticalement et 2 cellule horizontalement*).

Pour l'expansion du label:

- De nouveau, nous utilisons **sticky="EW"** pour **Tkinter**
- Et nous utilisons **wx.EXPAND** pour **wxPython**.

✱ **A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !**

Ok. 3 widgets. Et ensuite ?

Etape 12 : Activer le redimensionnement automatique

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = Tkinter.Button(self,text=u"Click me !")
        button.grid(column=1,row=0)

        label = Tkinter.Label(self,
                               anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
```

```

        self.grid_columnconfigure(0,weight=1)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowbleCol(0)
        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Maintenant nous allons demander à notre gestionnaire de layout de changer la taille de ses colonnes et lignes quand la fenêtre est redimensionnée.

Enfin... pas les lignes. Et seulement la première colonne (0).

C'est **AddGrowableCol()** pour **wxPython**.

C'est **grid_columnconfigure()** pour **Tkinter**.

Notez qu'il y a des paramètres supplémentaires. Par exemple, certaines colonnes peuvent s'agrandir plus que d'autre. C'est à ça que sert le paramètre **weight**: partager l'espace disponible dans des proportions différentes pour les différentes colonnes/lignes. (Dans notre cas, on ne fait rien de spécial: On met 1).

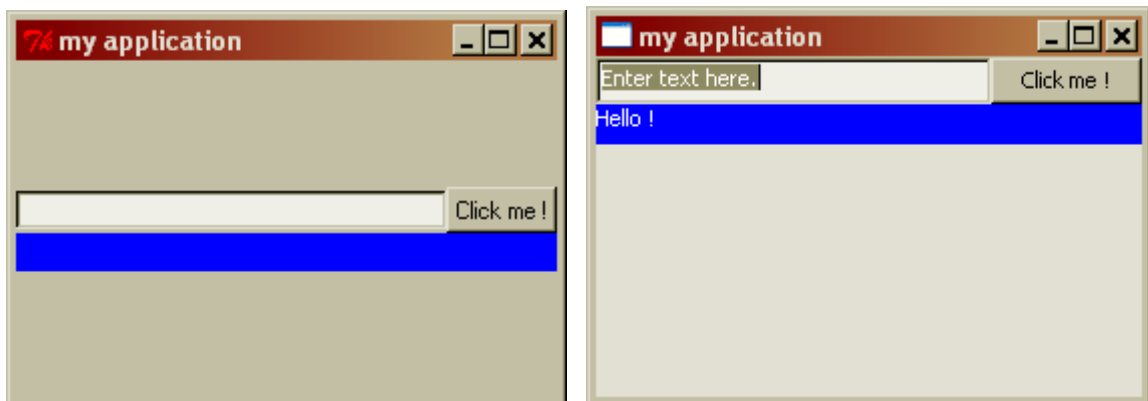
✳ ***A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !***

Maintenant essayez de redimensionner la fenêtre.

Vous voyez ?

Le champ texte et le label bleu se redimensionnent automatiquement en conséquences pour s'adapter à la largeur de la fenêtre.

Ceci dit, quand on agrandit la fenêtre **verticalement**, ça n'est pas très beau:



Ajoutons donc une contrainte pour que l'utilisateur ne puisse agrandir la fenêtre **que** horizontalement.

Etape 13 : Ajouter une contrainte

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = Tkinter.Button(self,text=u"Click me !")
        button.grid(column=1,row=0)

        label = Tkinter.Label(self,
                               anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

```

```

def initialize(self):
    sizer = wx.GridBagSizer()

    self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
    sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

    button = wx.Button(self,-1,label="Click me !")
    sizer.Add(button, (0,1))

    self.label = wx.StaticText(self,-1,label=u'Hello !')
    self.label.SetBackgroundColour(wx.BLUE)
    self.label.SetForegroundColour(wx.WHITE)
    sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

    sizer.AddGrowableCol(0)
    self.SetSizerAndFit(sizer)
    self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
    self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Pour empêcher la fenêtre d'être redimensionnée verticalement:

- Avec **Tkinter**, on utilise **.resizable(True,False)**.
- Avec **wxPython**, vous pouvez spécifier les hauteurs et largeurs minimales et maximales de votre fenêtre.
 Nous réglons les hauteurs minimales et maximales à la hauteur actuelle de notre fenêtre (**self.GetSize().y**) de manière ce qu'il ne soit pas possible de la redimensionner verticalement.
 Nous laissons **-1, -1** pour la largeur afin que la fenêtre puisse être librement redimensionnée horizontalement (**-1** veut dire "pas de limite").

✱ *A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !*

Essayez maintenant de redimensionner la fenêtre.

Etape 14 : Ajouter des gestionnaires d'évènements

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)

        button = Tkinter.Button(self,text=u"Click me !",
                                command=self.OnButtonClick)
        button.grid(column=1,row=0)

        label = Tkinter.Label(self,
                               anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)

    def OnButtonClick(self):
        print "You clicked the button !"

    def OnPressEnter(self,event):
        print "You pressed enter !"

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx

```



```

except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowablesCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
        self.Show(True)

    def OnButtonClick(self,event):
        print "You clicked the button !"

    def OnPressEnter(self,event):
        print "You pressed enter !"

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Les gestionnaires d'évènements sont des méthodes qui sont appelées quand quelque chose se passe

dans l'interface graphique (bouton cliqué, etc.)

Nous allons lier ces gestionnaires d'évènement à des **widgets spécifiques** pour des **événements spécifiques**.

Nous allons donc faire quelque chose quand le bouton est cliqué, ou la touche ENTREE est pressée dans le champ texte.

- On crée une méthode **OnButtonClick()** qui sera appelée quand le bouton est cliqué.
- On crée une méthode **OnPressEnter()** qui sera appelée quand la touche ENTREE sera pressée dans le champ texte.

On lie ("*bind*") ensuite ces méthodes aux widgets:

Pour le **bouton**:

- Avec **Tkinter**, on ajoute simplement **command=self.OnButtonClick** au bouton.
- Avec **wxPython**, on utilise la méthode **.Bind()**:
button est le widget sur lequel on veut "attraper" un évènement (notre bouton).
wx.EVT_BUTTON est le type d'évènement qu'on veut attraper (clic sur un bouton).
self.OnButtonClick est la méthode qui sera appelée si le bouton est cliqué.

Pour le **champ texte**:

- Avec **Tkinter**, on utilise la méthode **.bind()**
"**<Return>**" est la touche que nous voulons "attraper".
self.OnPressEnter est la méthode qui sera appelée si cet évènement se produit.
- Avec **wxPython**, c'est à nouveau la méthode **.Bind()** que nous utilisons, mais cette fois sur l'évènement **wx.EVT_TEXT_ENTER**.

Ainsi:

- Cliquer sur le bouton appellera la méthode **OnButtonClick()**.
- Presser ENTREE dans le champs texte appellera la méthode **OnPressEnter()**.

 **A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !**

Entrez un peu de texte, et pressez ENTREE ou cliquez sur le bouton: Du texte apparaîtra.
(Tkinter l'affichera dans la console (ou fenêtre MS-Dos) ; wxPython affichera une popup.)

Etape 15 : Modifier le label

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)

        button = Tkinter.Button(self,text=u"Click me !",
                                command=self.OnButtonClick)
        button.grid(column=1,row=0)

        self.labelVariable = Tkinter.StringVar()
        label = Tkinter.Label(self,textvariable=self.labelVariable,
                              anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)

    def OnButtonClick(self):
        self.labelVariable.set("You clicked the button !")

    def OnPressEnter(self,event):
        self.labelVariable.set("You pressed enter !")

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```

try:
    import wx
except ImportError:
    raise ImportError, "The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self, -1, value=u"Enter text here.")
        sizer.Add(self.entry, (0,0), (1,1), wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self, -1, label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self, -1, label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0), (1,2), wx.EXPAND )

        sizer.AddGrowCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1, self.GetSize().y, -1, self.GetSize().y );
        self.Show(True)

    def OnButtonClick(self, event):
        self.label.SetLabel("You clicked the button !")

    def OnPressEnter(self, event):
        self.label.SetLabel("You pressed enter !")

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None, -1, 'my application')
    app.MainLoop()

```

Maintenant modifions le label bleu:

- Avec **wxPython**, c'est facile: on appelle simplement la méthode **.SetLabel()** sur notre label (**self.label**)
- Avec **Tkinter**, c'est un peu plus compliqué. Vous devez:
 - - créer une variable spéciale Tkinter qui contiendra la valeur (**self.labelVariable = Tkinter.StringVar()**)
 - lier cette variable au widget (**textvariable=self.labelVariable**)
 - puis utiliser les méthodes **set()** or **get()** sur la variable Tkinter pour modifier ou lire sa valeur (**self.labelVariable.set("You clicked the button !")**)

Avec Tkinter, chaque fois que vous voulez lire ou modifier une valeur dans un widget (champ texte, label, case à cocher, bouton radio...), vous devez créer une variable Tkinter et la lier au widget. Il existe plusieurs types de variables Tkinter (StringVar, IntVar, DoubleVar, BooleanVar).

Par exemple, on utilisera typiquement une variable Tkinter BooleanVar pour une case à cocher.

Avec wxPython, vous appelez directement une méthode pour lire/modifier la valeur du widget.

✿ *A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !*

Pressez ENTREE ou cliquez sur le bouton: Le label bleu est modifié.

Etape 16 : Afficher la valeur

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        self.grid()
```

```
        self.entryVariable = Tkinter.StringVar()
```

```
        self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
```

```
        self.entry.grid(column=0,row=0,sticky='EW')
```

```

self.entry.bind("<Return>", self.OnPressEnter)
self.entryVariable.set(u"Enter text here.")

button = Tkinter.Button(self,text=u"Click me !",
                        command=self.OnButtonClick)
button.grid(column=1,row=0)

self.labelVariable = Tkinter.StringVar()
label = Tkinter.Label(self,textvariable=self.labelVariable,
                      anchor="w",fg="white",bg="blue")
label.grid(column=0,row=1,columnspan=2,sticky='EW')
self.labelVariable.set(u"Hello !")

self.grid_columnconfigure(0,weight=1)
self.resizable(True,False)

def OnButtonClick(self):
    self.labelVariable.set( self.entryVariable.get()+ " (You clicked the button)" )

def OnPressEnter(self,event):
    self.labelVariable.set( self.entryVariable.get()+ " (You pressed ENTER)" )

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")

```

```

sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

button = wx.Button(self,-1,label="Click me !")
sizer.Add(button, (0,1))
self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

self.label = wx.StaticText(self,-1,label=u'Hello !')
self.label.SetBackgroundColour(wx.BLUE)
self.label.SetForegroundColour(wx.WHITE)
sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

sizer.AddGrowbleCol(0)
self.SetSizerAndFit(sizer)
self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
self.Show(True)

def OnButtonClick(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )

def OnPressEnter(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Maintenant lisons la valeur du champ texte pour l'afficher dans le label bleu:

- Avec **wxPython**, on appelle **self.entry.GetValue()**
- Avec **Tkinter**, nous devons à nouveau créer une variable Tkinter dont nous lirons la valeur avec **.get()**.

Comme nous avons maintenant une variable Tkinter pour accéder au texte du champs et au label, on peut également mettre les valeurs par défaut ("Enter text here." et "Hello !")

 **A cet instant, vous pouvez lancer les deux programmes: Ils fonctionnent !**

Entrez un peu de texte dans le champ, et pressez ENTREE ou cliquez sur le bouton: Le label bleu va afficher le texte que vous avez tapé.

Etape 17 : Petit raffinement

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entryVariable = Tkinter.StringVar()
        self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)
        self.entryVariable.set(u"Enter text here.")

        button = Tkinter.Button(self,text=u"Click me !",
                                command=self.OnButtonClick)
        button.grid(column=1,row=0)

        self.labelVariable = Tkinter.StringVar()
        label = Tkinter.Label(self,textvariable=self.labelVariable,
                              anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
        self.labelVariable.set(u"Hello !")

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)
        self.entry.focus_set()
        self.entry.selection_range(0, Tkinter.END)

    def OnButtonClick(self):
        self.labelVariable.set( self.entryVariable.get()+" (You clicked the button)" )
        self.entry.focus_set()
        self.entry.selection_range(0, Tkinter.END)
```



```

def OnPressEnter(self,event):
    self.labelVariable.set( self.entryVariable.get()+" (You pressed ENTER)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowableCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
        self.entry.SetFocus()

```

```

self.entry.SetSelection(-1,-1)
self.Show(True)

def OnButtonClick(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )
    self.entry.SetFocus()
    self.entry.SetSelection(-1,-1)

def OnPressEnter(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )
    self.entry.SetFocus()
    self.entry.SetSelection(-1,-1)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Voici un petit raffinement: Le champ texte sera automatiquement re-sélectionné après que l'utilisateur ait pressé ENTREE ou cliqué sur le bouton. Il pourra ainsi taper immédiatement un nouveau texte dans le champ (en remplaçant le texte existant).

Nous commençons par donner le focus au champ texte (**focus_set()** et **SetFocus()**), puis nous sélectionnons le texte (**selection_range()** et **SetSelection()**).

Etape 18 : Correction d'un comportement de Tkinter

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

```

```

def initialize(self):
    self.grid()

    self.entryVariable = Tkinter.StringVar()
    self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
    self.entry.grid(column=0,row=0,sticky='EW')
    self.entry.bind("<Return>", self.OnPressEnter)
    self.entryVariable.set(u"Enter text here.")

    button = Tkinter.Button(self,text=u"Click me !",
                           command=self.OnButtonClick)
    button.grid(column=1,row=0)

    self.labelVariable = Tkinter.StringVar()
    label = Tkinter.Label(self,textvariable=self.labelVariable,
                          anchor="w",fg="white",bg="blue")
    label.grid(column=0,row=1,columnspan=2,sticky='EW')
    self.labelVariable.set(u"Hello !")

    self.grid_columnconfigure(0,weight=1)
    self.resizable(True,False)
    self.update()
    self.geometry(self.geometry())
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

def OnButtonClick(self):
    self.labelVariable.set( self.entryVariable.get()+" (You clicked the button)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

def OnPressEnter(self,event):
    self.labelVariable.set( self.entryVariable.get()+" (You pressed ENTER)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:

```

```

import wx
except ImportError:
    raise ImportError, "The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self, -1, value=u"Enter text here.")
        sizer.Add(self.entry, (0,0), (1,1), wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self, -1, label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self, -1, label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0), (1,2), wx.EXPAND )

        sizer.AddGrowablesCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1, self.GetSize().y, -1, self.GetSize().y );
        self.entry.SetFocus()
        self.entry.SetSelection(-1, -1)
        self.Show(True)

    def OnButtonClick(self, event):
        self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )
        self.entry.SetFocus()
        self.entry.SetSelection(-1, -1)

    def OnPressEnter(self, event):
        self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )
        self.entry.SetFocus()
        self.entry.SetSelection(-1, -1)

if __name__ == "__main__":

```

```
app = wx.App()
frame = simpleapp_wx(None,-1,'my application')
app.MainLoop()
```

Tkinter a une particularité: Il essaie constamment d'adapter la taille de la fenêtre à ce qu'elle contient. Ça part d'un bon sentiment, mais ce n'est pas toujours ce que l'on souhaite. Par exemple, entrez un loooooooooooooooooooooong texte dans le champ, et pressez ENTREE: La fenêtre d'agrandit.

Entrez à nouveau un texte court et pressez ENTREE: La fenêtre rétrécit.

Généralement, on ne veut pas que la fenêtre passe son temps à changer de taille toute seule. Les utilisateurs n'aiment pas ça.

C'est pour cela que nous figeons la taille de la fenêtre à sa propre taille (**self.geometry(self.geometry())**).

De cette manière, Tkinter cessera d'essayer tout le temps d'adapter la fenêtre à son contenu.

(Et nous faisons un **update()** pour être sûr que Tkinter a terminé le rendu des différents widgets qu'il contient et qu'il a terminé de déterminer leur taille.)

wxPython/wxWidgets n'a pas ce comportement, nous n'avons donc rien à faire de spécial.

C'est terminé !

Nous avons maintenant notre application. :-)

Vous pouvez télécharger les sources des deux programmes: [simpleapp_tk.py](#) [simpleapp_wx.py](#)

Maintenant, quelques remarques.

Outils RAD et coordonnées pixel

Il peut paraître lourd de construire des interfaces graphiques de cette manière, mais quand on a compris le principe, il est assez simple d'ajouter des éléments et de les assembler dans des conteneurs (onglets, panneau redimensionnables, scrollables...). Au final, c'est très souple.

Si vous préférez les environnement RAD dans le genre de VB (VisualBasic), vous pouvez vous tourner vers *Boa Constructor* (conçu pour wxPython). Mais j'apprécie de moins en moins de genre d'outils.

Ils tendent à générer du code inutile, lourd, et certains environnements n'arrivent parfois même pas à relire le code (Est-ce qu'il y a des utilisateurs de VisualStudio.Net dans la salle ? Je n'aime pas que le designer me vienne complètement modifier mon interface, qu'il me fasse disparaître sans raison des événements, qu'il ne comprenne pas les bases du HTML et CSS ou que le designer me fasse une faute de protection générale en ré-ouvrant un projet. :-@).

Créer une interface graphique sans un RAD est un peu plus long, mais vous avez un bien meilleur contrôle sur ce qui est fait, et surtout **vous pouvez utiliser des gestionnaire de layout de type grille (grid)**. (La plupart des environnements RAD ne travaillent qu'en coordonnées pixel fixes.)

Pourquoi éviter les coordonnées fixes en pixels ?

- Est-ce que vous avez déjà rencontré ces logiciels où [une partie du texte est illisible parcequ'il dépasse d'un widget](#) ? Ce sont les coordonnées pixel en action !
Si vous utilisez les coordonnées pixels, votre interface graphique risque d'être inutilisable avec des polices de taille différentes.
Avec un gestionnaire de type *grid*, les widgets s'adaptent.
- Vous n'avez jamais été énervé par ces logiciels avec [une taille de fenêtre ridiculement minuscule](#) alors que vous avez un écran 1600x1200 ? C'est encore les coordonnées pixel en action.
Avec un gestionnaire de type *grid*, les utilisateurs peuvent redimensionner la fenêtre pour profiter de votre logiciel en plein écran.

Interface graphique non bloquante ?

Vous devez garder à l'esprit que lorsqu'un gestionnaire d'évènement est en cours d'exécution, la boucle d'évènement du toolkit ne tourne plus et ne traite plus les évènements qui arrivent (Les nouveaux évènements sont alors placés dans une file d'attente.)

Donc l'interface graphique est complètement bloquée tant que le gestionnaire d'évènement n'a pas terminé son travail.

(Les boutons et menus ne répondent pas, et la fenêtre ne peut même pas être déplacée ou redimensionnée. Je suis certain que vous avez déjà rencontré ce genre de logiciel.)

Vous avez deux solutions à ce problème:

- Effectuer seulement des actions *courtes* dans les gestionnaires d'évènement.
- Utiliser les threads.

La programmation par multi-threads n'est pas triviale. Elle peut même devenir un cauchemard à déboguer. Mais c'est le prix à payer si vous voulez créer un logiciel qui répond bien aux sollicitations de l'utilisateur. (Par exemple, mon programme [webGobbler](#) possède un thread dédié exclusivement à l'interface graphique, ce qui fait que le logiciel répond bien à l'utilisateur même quand il est en train de faire des traitements sur les images ou qu'il attend la réponse de requêtes réseau.)

La plupart des utilisateurs attendent des logiciels récents qu'il soient non bloquants.

Attention: La plupart des toolkits graphique ne sont pas thread-safe. Cela veut dire que si deux threads essaient de modifier la valeur d'un même widget, cela peut planter toute votre application (et même générer une faute de protection générale). Vous devez utiliser des sections critiques et d'autres moyens (queues de messages, etc.)

Python supporte les threads et fournit différents objets pour régler ces problèmes (classe `Queue` thread-safe, sémaphores, sections critiques...)


Séparer nettement la logique de votre programme de l'interface graphique facilitera grandement le passage au mode multi-threads.


C'est pour cela que je recommande de séparer l'interface graphique et la logique du programme dans ce document.

De plus, cela vous permettra même de créer plusieurs interfaces graphiques pour votre programme si vous le souhaitez !


Tkinter ou wxPython ?


Tkinter / Tcl/tk :


 Tkinter est fourni en standard avec Python. La plupart des utilisateurs de Python pourront utiliser votre programme tel quel.

 Pas de widgets avancés, même si on peut palier cela avec [Pmw](#) (Python Megawidgets).

wxPython / wxWidgets:

 Utilise l'aspect natif du système d'exploitation quand possible (aspect graphique plus proche de celui du système d'exploitation).

 Widgets avancés (sélecteur de date, bar d'outils flottantes, fenêtres non-rectangulaires, arbres, bulles d'aide...)

 Ne fait pas partie de la distribution standard de Python. Doit être téléchargé et installé séparément.

Autres toolkit graphiques

Il existe beaucoup d'autres toolkits accessibles en Python, comme GTK (à travers pyGTK), Qt (à travers PyQt), MFC, SWING, Fltk et même des wrappers autour de Tkinter/wxPython eux-mêmes (PythonCard, etc.).

Pour d'autres toolkits, voir:

- <http://wiki.python.org/moin/GuiProgramming>
- http://home.pacbell.net/atai/guitool/#free_python
- <http://awaretek.com/toolkits.html>

Créer un fichier EXE

Si votre application utilise Tkinter ou wxPython, il est possible de la packager sous forme d'exécutable (binaire) avec des programmes comme py2exe ou cx_Freeze.

(Même si parfois cela nécessite quelques bidouillages comme celui-là:

http://sebsauvage.net/python/snypets/index.html#tkinter_cxfreeze)

Voir (en anglais): <http://sebsauvage.net/python/snypets/index.html#py2exe>

Par exemple, [webGobbler](#) utilise Tkinter et a été packagé avec cx_Freeze et InnoSetup sous forme d'un installateur pour Windows.

A propos de ce document

Ce document a été écrit par Sébastien SAUVAGE, webmaster de <http://sebsauvage.net>

L'adresse de ce document est http://sebsauvage.net/python/gui/index_fr.html

C'est la traduction française du document: <http://sebsauvage.net/python/gui/index.html>

Dernière mise à jour: 2006-10-25.