INTRODUCTION À WXPYTHON

Jeremy Berthet & Gilles Doge
HEIG-VD, le 7 juillet 2006

Table des matières

1. Introduction

Le document que vous avez entre les mains a été écrit initialement à la HEIG-VD dans le cadre d'un projet de trimestre.

Son objectif est de vous présenter la librairie wxPython[2] en decrivant son fonctionnement ainsi que les fonctionnalités qu'elle met à disposition.

Dans l'idéal, ce document devrait pouvoir accompagner un cours d'interface utilisateur, tel que celui enseigné en 2ème année de la voie "Informatique Logiciel" de l'HEIG-VD, en utilisant Python et wxPython comme outil de démonstration des concepts enseignés.

1.1. Présentation générale

L'API wxPython permet la création d'interface utilisateur basique et évoluée. Elle offre les différents objets que l'on peut s'attendre à rencontrer dans une application moderne, la possibilité de dessiner en deux dimensions, et supporte des interfaces de type MDI (Multiple Document Interface).

wxPython est une API simple d'utilisation prévue pour être utilisée avec le langage Python. Il s'agit d'une adaptation d'une autre API à l'origine prévue pour le langage C++ nommée wxWidgets[3].

Ce dernier point implique que la structure de l'API (hiérarchie des classes, nom des méthodes, etc.) est presque exactement la même pour les deux et qu'il est donc possible de se référer aisément aux nombreux documents existant pour wxWidgets.

2. Installation

2.1. Linux/Unix

Pour la majorité des distributions Linux, il existe des paquets qui feront tout pour vous. Des RPM sont disponibles sur le site de wxPython et les paquet `libwxgtkX.Y-python` sont présents dans les dépôt Debian (X et Y étant les numéros de version).

Autrement, wxPython a comme dépendances les librairies "glib" et "gtk+" qui sont installée par défaut dans un environnement GNOME.

2.2. Windows

En vous rendant sur le site wxPython[2], vous trouverez un programme d'installation de la librairie pour Python 2.3, 2.4 et en ANSI ou unicode.

2.3. MacOS 9 et MacOS X

La librairie wxPython est installée par défaut depuis MacOS X 10.3 (Panther) (mais pour Python 2.3 néanmoins) et ne nécessite donc aucune installation spéciale. A la différence d'un programme python normal, il faudra lancer vos script wxPython avec l'interpreteur `pythonw`. Si vous posséder une version antérieure de MacOS, vous devrez installer wxPython a partir des paquetages externe[5].

3. Concepts de base

3.1. Hello World!

L'apprentissage commence généralement par l'exemple le plus simple possible. Nous ne ferons pas exception à la règle. Vous pouvez ouvrir le premier exemple intitulé "hello.py" dans votre éditeur préféré et l'exécuter.

Vous devriez obtenir une présentation semblable à cette dernière (il peut y avoir des variations suivant l'OS que vous utilisez).

Fig. 1: Hello World !

Si le programme refuse de s'exécuter, reportez-vous au chapitre concernant l'installation sur votre système et reportez-vous a la documentation.

Vous pouvez constater qu'il n'a pas fallu beaucoup de ligne de code pour faire cela. De plus, ce premier exemple est excessivement structuré en deux classes, ce qui n'est pas forcément obligatoire.

Analysons quelques lignes essentielles:

```
    import wx
```

Toute application wxPython possédera au moins cette demande d'import. Il s'agit d'importer les classes mises à dispositions par le module dans notre application afin de pouvoir les utiliser.

La classe MainWindow dérive de `wx.Frame`, la classe de base dans wxPython pour représenter une fenêtre à l'écran. Le constructeur de la classe appelle celui de son parent avec des paramètres dont vous pouvez déjà deviner l'utilité pour certains.

```
    self.Show(True)
```

La méthode Show est héritée de wx.Frame et permet de dire à la fenêtre de s'afficher ou pas à l'écran suivant le paramètre fourni.

wxPython utilise la classe particulière `wx.App` (plusieurs sont disponibles en réalité) pour définir l'application dans son ensemble. Cette encapsulation supplémentaire permet de gérer plus facilement certains concepts propres aux interfaces graphiques.

Dans notre exemple, la classe MainApp en dérive et crée l'objet représentant notre fenêtre principale. Pour des raisons de gestion propre à wxPython, il faut redéfinir la méthode OnInit à la place du constructeur. Cette méthode créer une instance de notre fenêtre et lui demande de s'afficher en premier plan.

Finalement, l'instance de notre application est créée uniquement si ce script est le programme principal. Nous expliquerons l'appel à MainLoop dans la partie concernant la gestion d'événements.

3.2. La classe `wx.App`

Au sous-chapitre précédent, nous avons vu un programme formé de deux classes. Ces deux classes ne sont pas là par hasard. Dans toutes application wxPython, les classe `App` et `Frame` sont les composants de base dont découle le reste.

Bien qu'il soit possible d'instancier directement des objets de ces deux classes, l'usage veut qu'on les dérive tout comme cela a été fait dans le premier programme d'exemple. La classe Frame sera décrite en détail dans le chapitre 6. Ici, nous nous développerons plus en détail la classe `App`.

Dans une application wxPython, il est nécessaire d'avoir une (et une seule uniquement) instance de App avant d'instancier le moindre composant graphique (à priori, la fenêtre principale). Effectivement, cet objet particulier interagit directement avec le système et c'est lui qui prépare le terrain pour la création de vos composants graphiques.

De ce fait, il est fortement déconseillé de redéfinir un constructeur pour les classes héritant de App, car vous risqueriez de perturber le fonctionnement interne. C'est pour cette raison que la classe App dispose d'une méthode `OnInit()` automatiquement invoquée une fois l'objet instancié.

Pour être sur de ne pas créer de composant graphique avant d'avoir créer un objet App, nous prendrons l'habitude de créer les instances de fenêtres dans la méthode `OnInit()`, d'où l'intérêt de toujours avoir une sous-classe de App, même si ses fonctionnalités ne sont pas étendues.

En plus de préparer le système à la création de composant graphique, l'objet `App` possède le gestionnaire d'événement qui est démarré lors d'un appel à la méthode `MainLoop()`. Basiquement, le gestionnaire d'événement est une boucle infinie qui fait deux choses.

En premier lieu, elle attend de recevoir des signaux de la part du système d'exploitation (un clic de souris par exemple) puis lorsqu'elle reçoit un tel signal, elle l'ajoute dans une file d'attente avec divers paramètres (type d'événement, source de l'événement, etc.).

Ensuite, elle retire périodiquement un événement de la queue et invoque la méthode liée selon les paramètres qu'elle peut lire. Si aucune méthode n'est liée, elle oublie l'événement et retire le

prochain de la file.

Du coup, tout ce que le programmeur doit faire, c'est lié une méthode à un événement comme nous le verrons dans le sous-chapitre suivant.

La classe App permet encore de faire plusieurs choses qui dépassent la portée de ce tutoriel. Vous pouvez vous référer au livre "wxPython in action"[1] (en Anglais) pour plus d'informations.

Notez finalement que le gestionnaire d'événement, bien que l'on dise que sa boucle est infinie, s'interrompt si plus aucune instance de Frame existe dans le programme. L'instance de `App` est alors détruite et vous ne pourrez plus recréer de composant graphique avant d'avoir recréer une telle instance.

3.3. Liaison widget/code

Un événement est généré quand "quelque chose se passe sur un objet wxPython". Ce "quelque chose" peut être de nature très différente. Il peut avoir été généré par l'utilisateur, lors d'un clic sur un bouton par exemple ou avoir été généré directement par un objet quand il a changé d'état.

Nous avons vu que les événements, une fois généré, étaient intercepté par un gestionnaire d'événement et que tout ce que nous avions à faire était de lié les actions à effectuer lorsque le gestionnaire détecte un événement précis. Le programme goodbye.py donne un bref exemple de réalisation.

Sans nous attarder sur la création d'un Panel et d'un bouton qui seront décrit plus tard, regardons la méthode clé, `Bind()`.

Cette méthode est définie dans la classe `EvtHandler` dont beaucoup de classe dérivent (`Frame` et `Menu` notamment, ainsi que `App` comme vous l'aurez sans doute deviné). Elle permet de faire une association entre un type d'événement, une méthode visible et un widget visible. Visible, dans le sens, de la visibilité du code et non pas du fait que l'objet est affiché ou pas à l'écran.

```
    self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
```

`wx.EVT_BUTTON` est une des nombreuses constantes à disposition représentants les événements qui peuvent survenir. Celle-ci représente un clic de souris sur un bouton de la frame (raison pour laquelle un appel au `Bind()` de la frame est fait). Au sein de la documentation online de wxPython, vous trouverez, pour les différents widgets, la liste des événements en explorant la documentation de la classe `wx.Event` et ses sous classes.

Dans cet exemple illustratif, nous avons une méthode `OnClick()` dans la classe `MainWindow`, qui demande à l'application de se fermer à l'aide de la fonction `Close()` qui sera expliquée plus tard.

Cette méthode pourrait porter n'importe quel nom, par exemple `ClickOnButton()`.

La méthode doit posséder un paramètre qui récupérera l'instance de l'événement. Ce paramètre doit être le deuxième dans le cas d'une méthode de classe (le premier étant `self`) ou le premier s'il s'agit d'une fonction "statique". Par convention, il s'appellera `event`.

```
    def OnClick(self, event):
```

Notez, qu'en tant que paramètre d'une autre méthode, nous passons l'instance de la méthode

`OnClick()` en paramètre, il ne faut donc pas mettre les parenthèses !

Finalement, le dernier paramètre est l'instance de l'objet source de l'événement à associer si l'appel de `Bind()` n'est pas fait directement sur l'objet provoquant l'événement comme dans ce cas (Le `EVT_BUTTON` est transmis depuis la frame, mais ne précise pas quel bouton a été cliqué).

Voilà, avec cette unique ligne, nous avons associé une méthode qui termine l'application au clic d'un bouton précis. Cette façon de faire est généralisable à tous les objets qui peuvent avoir des événements associés.

`Bind()` peut aussi prendre deux paramètres, id1 et id2 au lieu de l'objet source. Cela permet d'associer un même événements à une plage de plusieurs objets identifié par des id consécutifs. Néanmoins cette façon de faire est rarement utilisées. Il est préférable de manière générale de ne pas travailler avec les id. Pour information, chaque objet dans wxPython a une propriété id qui est un numéro unique.

Finalement, le paramètre source (et les paramètres id1, id2) est optionnel. A utiliser à vos risques et périls.

3.4. Propriétés communes des widgets

Tous les objets **affichables** dans wxPython dérivent d'une classe d'origine qui est `wx.Window`.

Le constructeur de cette classe se présente sous la forme suivante et permet donc de définir des propriétés communes à tous les objets affichables.

```
wx.Window(self, parent, id=-1, pos=DefaultPosition, size=DefaultSize, style=0,
          name=PanelNameStr)
```

Le paramètre `parent` définit, comme son nom l'indique, l'objet parent de celui que vous créez. Ceci permet de conserver une hiérarchie des objets qui sont ainsi contenu les uns dans les autres.

Si nous reprenons le code précédent, vous pouvez voir que la fenêtre (`wx.Frame`) contient un conteneur d'objet (`wx.Panel` pour lequel le parent est l'instance de `wx.Frame`) qui lui même contient un bouton.

Il est évidement possible de naviguer dans l'arbre ainsi crée à l'aide des méthodes suivantes:

```
GetParent()
```

Permet d'obtenir une référence sur l'objet parent.

```
GetChildren()
```

Permet d'obtenir une liste des références sur les enfants de l'objet. Il est important de savoir que cette liste est une copie de la liste maintenue par wxPython et donc, que la création d'un nouvel enfant ne la met pas à jours.

Cela peut révéler un intérêt pour certaine application ou dans le positionement à la volée des objets dans un panel. A vous de vous montrer créatifs.

En plus de la structure arborescente, wxPython entretient un numéro d'identification UNIQUE pour chaque widget. Avoir deux objets contenant le même ID peut se révéler désastreux et

amener l'application à des comportements inattendus.

Du coup, que mettre comme valeur pour la paramètre id ?

Si vous ne comptez pas utiliser explicitement ce numéro, autant laisser la valeur par défaut et passer les paramètres du constructeurs par nom, ou alors utiliser la valeur `wx.ID_ANY` si les paramètres sont passé par position. wxPython se chargera alors de générer automatiquement une ID valable pour votre widget.

Sinon, il est possible de générer une id à l'aide de la fonction `wx.NewId()` pour enregistrer une ID unique dans une variable et la passer en paramètre du constructeur après coup.

Finalement, quelque soit la méthode employée, tous les widgets disposent d'une méthode `GetId()` si le besoin s'en fait sentir.

Les paramètres `pos` et `size` du constructeur spécifient la position et la taille de départ de l'objet. Les constantes `DefaultPosition` et `DefaultSize` dépendent de l'objet qui est en train d'être créer.

Bien que wxPython définisse deux classe `wx.Point` et `wx.Size` avec les méthodes
appropriée, il est généralement plus simple d'utiliser un tuple de deux entier pour spécifier la position ou la taille d'un objet. Vous pouvez le constater dans l'exemple de code goodbye.py.

`wx.Window`, mais aussi de nombreux widgets en dérivant, définissent des styles qui peuvent influencer l'apparence des objets à l'écran. Ce sont des constantes listées dans le manuel de référence pour chaque widget.

Ces constantes définissent un style atomique et si vous voulez mélanger plusieurs style, vous pouvez utiliser l'opérateur | (OU logique). L'opération inverse est aussi possible, à l'aide de l'opérateur ^ qui retire des éléments de style à un mélange de styles atomiques.

Toujours dans notre exemple, goodbye.py, vous pouvez voir un exemple d'utilisation des opérateurs à la création de la fenêtre.

Attention tout de même à deux choses.
Le manuel de référence est encore loin d'être complet et une recherche sur internet vous informera parfois mieux sur les styles à disposition. Le livre "wxPython in action"[1] donne aussi de bonne astuces.

Les paramètres peuvent avoir des effets très différents en fonction de la plateforme d'exploitation. Ceci peut être gênant lors de l'utilisation du paramètre `pos` et nous verrons par la suite une méthode évoluée de positionnement.

Finalement, le paramètre `name` permet de donner un nom à l'objet et ne s'avère pas spécialement utile.

Vous connaissez désormais les bases du fonctionnement de wxPython. Par la suite, nous allons voir les différents objets que vous pourrez utiliser dans vos applications graphiques.

4. Fenêtrage

4.1. Frame et notion de Top-Level Window

Une interface graphique en wxPython contient au minimum un objet nommé «top-level window object» qui est généralement un objet de type `wx.Frame` (fenêtre). Cet objet n'est pas contenu dans un autre widget, c'est en quelque sorte la "racine" de l'application graphique. C'est généralement cet objet qui est défini comme la fenêtre principale de votre application et qui contient les autres objets graphiques.

Vous pouvez avoir plusieurs objets sans parent (plusieurs «top-levels window»), toutefois vous devez préciser à wxPython lequel sera considéré comme le top-levels window principal. Cette opération s'effectue grâce à la méthode `SetTopWindow()`. Si vous ne spécifiez pas explicitement quelle frame sera la "top-levels window" principale, wxPython prendra la première frame définie dans votre `wx.App`.

4.2. Quitter l'application

L'application quitte une fois que toutes les fenêtres "top-level" sont fermées, c'est à dire, une fois que toutes les frames sans parent sont fermées et pas seulement la frame désignée par la méthode `SetTopWindow()`.

Il est possible de fermer une frame en appelant la méthode `Close()`. Une autre façon de quitter l'application est d'utiliser la fonction globale `wx.Exit()`.

Il est possible de définir un traitement de «nettoyage» juste avant que l'application quitte grâce à la méthode `OnExit()` de la classe `wx.App`. En effet, cette méthode est appelée lorsque la dernière Frame principale est fermée. Ceci peut être pratique pour fermer des connexions réseau ou à une base de données par exemple.

4.3. Positionnement

Il existe 2 approches différentes pour disposer les éléments graphiques dans une fenêtre. La première, utilise le positionnement absolu tandis que la deuxième utilise un élément spécial nommé «Sizer».

4.3.1. Absolu

Le positionnement absolu est un procédé très simple et intuitif pour placer les différents objets. Il suffit d'utiliser des coordonnées (X, Y) où l'origine se trouve en haut à gauche la frame. Cette méthode va très bien quand on possède un contrôle total sur la taille de la fenêtre et sur le nombre de widget la contenant. Toutefois, on arrive vite aux limites de cette approche, notamment lorsque la taille de la fenêtre n'est pas fixe. Il est dans ce cas, préférable d'utiliser les Sizers.

4.4. Sizers

4.4.1. Positionnement à l'aide de Sizers

Un Sizer est un objet dont le seul but est de gérer la disposition d'un jeu de widgets à l'intérieur d'un conteneur. Le Sizer n'est pas un conteneur ou un widget en soit. C'est en réalité une représentation d'un algorithme de positionnement. Tout les Sizers sont des instances d'une sous-classe de la classe abstraite `wx.Sizer`.

Il y cinq sizers fourni par wxPython :

Type de Sizer	Description
Grid	Une simple grille régulière (toutes les cellules ont la même taille). Peu typiquement être utilisé pour un plateau de jeu simple.
Flex Grid	Une grille flexible, permet de meilleur rendu lorsque les widgets n'ont pas la même taille.
Grid Bag	Le sizer le plus flexible. Permets un placement arbitraire des widgets à l'intérieur de la grille.
Box	Utile lors de positionnement de widgets de manière horizontale ou verticale.
Static Box	Permet d'entourer d'une bordure un groupe de widget. Il est aussi possible d'y joindre un texte.

L'utilisation d'un sizer s'effectue en 3 étapes :

1. Création du sizer dans un conteneur. Un sizer est appliqué à un widget conteneur en utilisant la méthode `SetSizer(sizer)` de l'objet `wx.Window` (la super-classe de tous les widgets affichable).
2. Ajouter chaque widget enfant au sizer, via la méthode `Add()` de `wx.Sizer`.
3. (Optionnel) activer le calcul automatique de la taille du sizer (appel en cascade de la méthode `Fit(window)` de `wx.Window`)

4.4.2. Ajouter des widgets au sizer

Pour ajouter des widgets au sizer, la solution la plus commune est d'utiliser la méthode `Add()`. Elle ajoute le widget à la **fin** de la liste de widgets enfant du Sizer. La méthode Add() est déclinée en 3 version:

```
Add(window, proportion=0, flag=0, border=0, userData=None)
Add(sizer,  proportion=0, flag=0, border=0, userData=None)
Add(size,   proportion=0, flag=0, border=0, userData=None)
```

La première est la plus utilisée, elle permet d'ajouter un widget (argument window) au sizer.
La seconde permet d'imbriquer les sizers.
La troisième est utilisée pour ajouter un espace vide dont la taille est définie par l'argument size (de type `wx.Size`).

Les autres paramètres influencent la manière dont le widget sera disposé. Plusieurs de ses paramètres sont disponible que pour certain Sizer. L'argument `proportion` est seulement utilisé par le box sizer, est permet de spécifié comment l'élément sera redimensionnable quand le conteneur parent changera de taille.
L'argument `flag` permet de spécifié des options d'affichage (comme l'alignement, la bordure ou encore le redimensionnement). L'argument `border` détermine la taille de la bordure si une bordure a été spécifié par le paramètres `flag`. Pour finir le paramètre `userData` peut être utilisé dans le cas ou vous définiriez votre propre Sizer.
Les différentes constantes à passer au paramètre flag se trouve dans la documentation de l'API sous la description de la méthode Add de wx.Sizer.

Il existe d'autres méthodes pour ajouter des widgets au Sizer, la méthode `Insert()` permet

d'ajouter un widget en spécifiant un index de positionnement, et la méthode `Prepend()`, identique à `Add()` mais qui ajoute cette fois ci, le widget en début de liste. Pour plus d'information, reportez-vous a la documentation de wxPython.

4.4.3. Supprimer des widget au Sizer

La méthode Detach() permet de supprimer un widget à un Sizer. Attention cette méthode ne détruit pas l'objet!

Elle a aussi trois déclinaison:

```
Detach(window)
Detach(sizer)
Detach(index)
```

Dans les trois cas, cette méthode retourne un booléen, elle retourne faux si vous essayé de supprimer un élément qui n'est pas dans le Sizer.
Respectivement, la 1ère déclinaison permet de supprimer un widget, la 2ème un sizer et la troisième permet de supprimer l'élément contenu a l'indice passé en paramètre.

Supprimer un élément du sizer ne changerais pas l'affichage automatiquement. Il faut pour cela appeler la méthode `Layout()`.

4.5. Le Grid Sizer

Le sizer le plus simple, comme son nom l'indique, il s'agit d'une grille bidimensionnel ou les widgets enfants sont placé de gauche à droite et de haut en bas. Il faut savoir que le sizer a en interne une liste des widgets enfants. Le premier widget de cette liste sera situé dans le coin en haut à gauche.

Lorsque que vous redimensionner un grid sizer, la taille de chaque cellule change de manière uniforme. Par défaut, cette taille se base sur le plus grand widget de la grille et la taille des widgets enfants ne s'adapte pas.

Le constructeur du Grid Sizer a la signature suivante :

```
wx.GridSizer(rows, cols, vgap, hgap)
```

Où :

- **rows** : le nombre de lignes de la grille ;
- **cols** : le nombre de colonnes de la grille ;
- **vgap** : l'espacement vertical (entre les colonnes) ;
- **hgap** : l'espacement horizontal (entre les lignes).

Si `row` ou `cols` a la valeur 0, le nombre de ligne ou de colonne dépendra du nombre de widgets dans le sizer. Par exemple, si le sizer a été créé avec les valeurs suivante `wx.GridSizer(2, 0, 0, 0)` et que le sizer contient 8 widgets, ils seront disposés en 2 lignes de 4 colonnes.

Les propriétés `rows`, `cols`, `vgap` et `hgap` ont tous des accesseurs : `GetRows()`, `SetRows(rows)`, `GetCols()`, `SetCols(cols)`, `GetVGap()`, `SetVGap(vgap)`, `GetHGap()`, `SetHGap(hgap)`.

4.6. Le Flex Grid Sizer

Ce sizer est plus flexible que le gridsizer vu ci-dessus. Voici les principales différences:

- Il détermine séparément la taille de chaque ligne et colonne;
- Par défaut, il ne change pas automatiquement la taille de ses cellules quand il est redimensionné. Vous pouvez spécifier quelles lignes et quelles colonnes seront automatiquement redimensionnées lors d'un changement de taille.

Le constructeur d'un Flex Grid Sizer a les mêmes paramètres que le gridsizer :

```
wx.FlexGridSizer(rows, cols, vgaps, hgap)
```

Pour définir quelles lignes ou colonnes pourront «grandir» en fonction de la taille du sizer, il faut passer par les méthodes suivantes :

```
AddGrowableCol(idx, proportion=0)
AddGrowableRow(idx, proportion=0)
```

Le premier paramètre définit la colonne ou la ligne qui doit devenir extensible (un entier entre 0 et N-1).

Lorsque le sizer grandi horizontalement, le fonctionnement par défaut fait que la nouvelle taille est reparti de manière équitable entre les colonnes définies comme extensible (via la méthode `AddGrowableCol()`). Il en ai de même pour les lignes lors d'un changement de taille vertical. Vous pouvez changer ce comportement par défaut en utilisant le paramètre proportion.

Par exemple, si vous avez défini 2 colonnes extensibles et que leurs proportions sont respectivement 2 et 1, la première colonne va prendre les 2/3 de l'espace disponible alors que la seconde ne prendra que 1/3.

Si vous précisez la proportion pour l'une de vos colonnes extensibles, vous devrez spécifier une proportion pour toutes vos colonnes. Il en va de même pour les lignes.

4.7. Le Grid Bag Sizer

Le grid bag sizer est une extension du flex grid sizer. Il apporte deux principales nouveautés :

- La possibilité d'ajouter un widget dans une cellule spécifique de grille;
- La possibilité de fusionner des lignes ou des colonnes.

Le constructeur d'un Grid Bag Sizer n'a pas besoin du nombre de lignes et de colonnes, car vous insérez vos objets en spécifiant directement un numéro de ligne et de colonne.
Sa signature est la suivante:

```
wx.FlexGridSizer(vgaps=0, hgap=0)
```

4.7.1. Ajouter un widget au wx.GridBagSizer

La méthode `Add()` du grid bag sizer est différente des autres sizer à cause des particularités vues ci-dessus.
Elle est déclinée en trois variantes, comme la méthode générique `Add()` de `wx.Sizer` :

```
Add(window, pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)
Add(sizer,  pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)
Add(size,   pos, span=wx.DefaultSpan, flag=0, border=0, userData=None)
```

Ce qui a changé, c'est l'apparition des paramètres `pos` et `span`.
Le paramètre `pos` permet de préciser où le widget va s'insérer. C'est un tuple (ligne, colonne) où la référence (0,0) est le coin supérieur gauche du sizer. Par exemple, pour insérer un objet dans la première ligne, troisième colonne il faut entrer le tuple (0,2).

Le paramètre `span` représente le nombre de colonnes et de lignes qui devront être fusionnées. Par exemple si vous souhaitiez que votre widget prenne 3 lignes et 2 colonnes, il faudra saisir le tuple (3,2).

Un exemple d'utilisation du grid bag sizer se trouve en annexes dans les exemples (gridbagsizer.py).

4.8. Box Sizer

Le box sizer est une unique colonne verticale ou une unique ligne horizontale, où les widgets sont disposé à la suite de haut en bas ou de gauche à droite.

Le constructeur n'a qu'un seul paramètre pour définir l'orientation :

```
wx.BoxSizer(orient)
```

orient peut prendre la valeur `wx.VERTICAL` (alignement vertical) ou `wx.HORIZONTAL` (alignement horizontal). Vous ne pouvez pas changer l'orientation du sizer par la suite, vous avez juste la possibilité de savoir dans quel sens il est orienté avec la méthode `GetOrientation()`.

4.9. Static Box Sizer

Le static box sizer combine un box sizer avec un objet static box. L'objet static box permet de regrouper des widgets de manière visuelle en entourant le groupe d'une border avec un texte. C'est la classe `wx.StaticBox` qui permet de créer cet objet. Les widgets appartenant a ce groupe ne peuvent pas être directement inclus dans un objet de type `wx.StaticBox`, il faut donc créer un sizer. Le sizer en question se nomme `wx.StaticBoxSizer`.

Petit exemple :

```
box = wx.StaticBox(self, -1, "Label du groupe")
sizer = wx.StaticBoxSizer(box, wx.VERTICAL)

btn1 = wx.Button(self, -1, "Bouton 1")
btn2 = wx.Button(self, -1, "Bouton 2")

sizer.Add(btn1, 0, 0)
sizer.Add(btn2, 0, 0)

self.SetSizer(sizer)
sizer.Fit(self)
```

On crée en premier lieu un objet `wx.StaticBox` que l'on va lier avec un sizer `wx.StaticBoxSizer`. Puis on rajoute des widgets au sizer.

4.10. Boîtes de dialogues

wxPython offre tout un éventail de boîtes de dialogue prédéfinies. Elles dérivent toutes de la même classe parente wx.Dialog.

4.10.1. Boîte de dialogue avec un simple message

La boîte de dialogue la plus simple pour communiquer avec l'utilisateur est `wx.MessageDialog`. C'est une simple boîte avec un message ou les boutons à afficher sont configurables. Le code suivant montre une simple fenêtre modale ou il faut répondre par oui ou non.

```
msgDlg = wx.MessageDialog (None, 'Mangez-vous beaucoup de pommes ?',
                           'Question 1', wx.YES_NO | wx.ICON_QUESTION)
reponse = msgDlg.ShowModal()
# traitement selon reponse ici...

# Destruction de la boite de dialogue
msgDlg.Destroy()
```

Le constructeur d'une boite de dialogue avec message a la signature suivante

```
wx.MessageDialog ( parent, message, caption="Message box",
                   style=wx.OK | wx.CANCEL, pos=wx.DefaultPosition )
```

Où parent est la fenêtre parente (None si la boîte de dialogue doit être une fenêtre Top-Level). Le `message` est la chaîne de caractère qui doit être affichée, et `caption` est le titre qui doit apparaître dans le titre de la fenêtre. Notez que sous MacOS X, la chaîne `caption` n'apparaît pas dans le titre de la fenêtre, mais s'affiche en gras en dessus du message. Le `style` permet de définir quels boutons apparaîtront dans la fenêtre modale et il est même possible d'afficher des icônes prédéfinies. Le dernier paramètre, `pos`, permet de définir ou sera positionné la boîte de dialogue sur l'écran.

La méthode `ShowModal()` affiche la fenêtre modale et retourne l'identifiant (de type int) du bouton cliqué. Lorsque la boîte de dialogue est affichée, les autres fenêtres de l'application sont désactivées. Cette méthode est la même pour toutes les boîtes de dialogues. Voici une liste non exhaustive des valeurs de retour : `wx.ID_YES`, `wx.ID_NO`, `wx.ID_CANCEL`, `wx.ID_OK`.

Les styles d'une fenêtre modale peuvent être les suivants : `wx.OK`, `wx.CANCEL`, `wx.YES_NO`, `wx.YES_DEFAULT`, `wx.NO_DEFAULT`, `wx.ICON_ERROR`, `wx.ICON_EXCLAMATION`, `wx.ICON_INFORMATION`, `wx.ICON_QUESTION`.
Ils peuvent être combinés selon le principe du bitmask (avec l'opérateur '|').

4.10.2. Boite de dialogue avec saisie

Une autre boite de dialogue typique est la fenêtre modale avec un champ de saisie. wxPython là propose avec la classe `wx.TextEntryDialog` :

```
wx.TextEntryDialog (parent, message, caption="", defaultValue, style, pos)
```

`defaultValue` est la valeur par défaut du champ de saisie.
Vous avez la possibilité d'accéder ou de modifier au champ de saisie via les méthodes `GetValue()` et `SetValue(value)`.

4.10.3. Boîte de dialogue avec choix unique

Cette boîte de dialogue permet de choisir une valeur parmi un lot de valeurs. Ce lot est représenté sous la forme d'une liste (listbox). C'est la classe `wx.SingleChoiceDialog` qui gère ceci:

```
wx.SingleChoiceDialog (parent, message, caption, choices, style, pos)
```

Les paramètres sont identiques à que pour la `wx.TextEntryDialog` a l'exception qu'au lieu de passer une chaîne de caractère, on passe ici une liste chaîne de caractère.

Exemple:

```
listeOS = ['Linux', 'MacOS X', 'Windows', 'Autres']
dlg = wx.SingleChoiceDialog (self, 'Quel est votre OS ?', 'Question 2', listeOS)

if dlg.ShowModal() == wx.ID_OK:
   nomOS = dlg.GetStringSelection()
   idOS  = dlg.GetSelection()
```

`GetStringSelection()` permet de récupérer la chaîne de caractère sélectionnée tandis que `GetSelection()` renvoie l'index de l'élément sélectionné. Vous pouvez aussi définir une sélection avec la méthode `SetSelection(index)`.

4.10.4. Boite de dialogue avec choix multiple

La boîte de dialogue précédente permettait de choisir qu'un seul élément de la liste, `wx.MutliChoiceDialog` permet d'en sélectionner plusieurs. Utilisez `GetSelections()` pour récupérer la liste des index des éléments sélectionnés et `SetSelections(selections)` pour la définir.

4.10.5. Autre boites de dialogues

La librairie wxPython propose plusieurs autres boîtes de dialogues, comme des boîtes pour ouvrir un fichier/dossier, pour choisir une couleur, pour choisir une police (typo), une boîte de dialogue pour faire des rechercher-remplacer, etc.

Dans le dossier d'exemple, vous trouverez une script python utilisant les fenêtres modales vues ici ainsi qu'un « sélectionneur de couleur ». Le script se nomme `dialog.py`.

5. Gestion du focus

Dans ce chapitre et le suivant, nous allons nous intéresser au programme exemple gestion_liste.py

Celui-ci démontre les possibilités de gestion du focus pour améliorer le suivi de l'utilisateur au sein d'un programme et utilise plusieurs widgets de base qui vous serons présenté dans le chapitre 8.

Ce programme est loin d'être parfait dans sa gestion du focus pour l'utilisateur et vous êtes invité à l'améliorer si vous désirez vous exercer.

5.1. Événements du focus

Chaque fois qu'un widget obtient le focus d'une manière ou d'une autre, il génère un événement `wx.EVT_SET_FOCUS` qui peut être traité selon vos besoins.

```
    self.lstListe.Bind(wx.EVT_SET_FOCUS, self.OnListSelect)
```

Dans l'exemple, c'est lorsque la liste obtient le focus (donc, qu'un élément a été sélectionné) que les boutons "Modifier" et "Supprimer" sont activé. Vous pouvez ainsi voir un exemple d'utilisation de la méthode `Bind()` sur un widget autre que `Frame`.

Par symétrie, un widget qui perd le focus génère un événement `wx.EVT_KILL_FOCUS`.

5.2. Focus manuel

Il peut être parfois intéressant de donner manuellement le focus à un widget donné, spécialement après certaines opérations dans le programme.

La classe `wx.Window` (et donc l'ensemble des contrôles qui en hérite) met à disposition la méthode `SetFocus()` qui donne le focus au widget y faisant appel et génère les événements d'obtention et de perte du focus vu plus haut.

5.3. Interdire le focus

Il est possible d'empêcher un widget d'obtenir le focus à l'aide de la méthode `Enable()` qui prend comme unique paramètre une valeur booléenne.

Le widget devient désactivé si cette dernière est mise à `False` et ne pourra donc plus obtenir le focus. Evidémenent, il est possible d'annuler cet état de fait en appelant la méthode avec la valeur `True`.

5.4. Navigation tabulaire

Elle n'est active par défaut qu'à l'intérieur d'un widget de type `wx.Panel` ou d'une `wx.Dialog`. Il existe des moyens de l'activer dans d'autres conteneur, mais nous ne traiterons pas cette possibilité dans le cadre de ce document.

C'est pour cette raison que le programme d'exemple possède un `wx.Panel` dans lequel est ajouté le sizer principal.

En principe, l'ordre de navigation se fait selon le même ordre de création des enfants du `wx.Panel` au sein du code Python.

Dans le cadre de création dynamique de widget ou lorsqu'il faut clarifier le code, il est possible de placer un widget respectivement avant ou après un autre dans l'ordre de navigation à l'aide des méthodes suivantes.

```
MoveBeforeInTabOrder(win)
MoveAfterInTabOrder(win)
```

Dans le programme exemple, le bouton "Enregistrer" doit obtenir le focus avant la liste lorsque l'utilisateur appuie sur la touche <tab>.

```
self.cmdEnregistrer.MoveBeforeInTabOrder(self.lstListe)
```

5.5. Touches mnémoniques

En premier lieu, il est possible de définir au sein d'un conteneur (`wx.Panel et wx.Dialog`) une commande par défaut (pour un bouton) qui est actionnée lors d'un appui sur la touche <enter>. Il suffit simplement de faire appel à la méthode `SetDefault()` du widget qui est censé recevoir cette fonctionnalité.

Dans le programme exemple

```
self.cmdEnregistrer.SetDefault()
```

Il est aussi possibe d'associer des touches mnémonique au diverses entrée du menu et **(sous Windows seulement)** aux boutons présent dans une frame.

Pour cela, il faut créer une liste de tuples de trois éléments chacun qui fera office de table d'association. Ces tuples seront, par polymorphisme, des `wx.AcceleratorEntry` et se présentent sous la forme `(flag, keycode, cmdID)`.

`flag` représente les touches spéciales (<alt>, <ctrl>, etc ...) que l'utilisateur devra actionner. Elles sont définies par les constantes `wx.ACCEL_ALT`, `wx.ACCEL_SHIFT`, `wx.ACCEL_CTRL` et `wx.ACCEL_NORMAL` et il est possible de les combiner en utilisant la technique du bitmask vu au chapitre 5.4

`keycode` est la touche à combiner avec les touches spéciales pour obtenir l'activation de la commande. Il s'agit aussi d'un entier qu'on peut déterminer en utilisant la fonction intégrée `ord()` en passant comme paramètre la lettre désirée.

`cmdID` finalement est l'id de l'entrée du menu ou du bouton à associer avec la touche mnémonique. C'est une première utilisation effective des identifiant unique dont nous avons brièvement parlé au chapitre 5.4

Le programme exemple contient un exemple de définition de table pour des boutons

```
table = [(wx.ACCEL_ALT, ord('E'), self.cmdEnregistrer.GetId()),
         (wx.ACCEL_ALT, ord('M'), self.cmdModifier.GetId()),
         (wx.ACCEL_ALT, ord('S'), self.cmdSupprimer.GetId()),
         (wx.ACCEL_ALT, ord('Q'), self.cmdQuitter.GetId())]
```

Une fois la table définie, il suffit d'instancier une `wx.AcceleratorTable` avec cette dernière

comme paramètre à la méthode `SetAcceleratorTable()` de la frame qui sera concernée par les touches définies

```
    self.SetAcceleratorTable(wx.AcceleratorTable(table))
```

Pour que la lettre concernée soit soulignée dans le label du contrôle, il suffit d'ajouter un '&' devant celle ci. Par exemple '&Enregistrer'.

Pour aller plus loin et améliorer le système, un article du wiki wxPython propose une solution de navigation par touche mnémonique (http://wiki.wxpython.org/index.cgi/FocusingControlsWithTheirLabels).

6. Widget principaux

Après avoir présenter l'aspect de gestion du focus dans le programme exemple gestion_liste.py, nous allons étudier de manière sommaire les différents widgets qui le composent.

Nous ne présenteront dans ce chapitre que les widgets de base et les fonctionnalités principales qui leur sont associés. Les possibilités de chaque widgets sont souvent bien plus grande et le wiki wxPython ainsi que la documentation officielle vous seront utile à ce niveau.

6.1. wx.StaticText

Souvent appelé "label" dans d'autres librairies d'interface utilisateur, ce widget permet d'afficher un simple texte, non modifiable par l'utilisateur à l'écran.

Le constructeur se présente sous la forme suivante

```
wx.StaticText(parent, id=-1, label=EmptyString, pos=DefaultPosition,
              size=DefaultSize, style=0, name=StaticTextNameStr)
```

Rien de bien nouveau par rapport au constructeur `wx.Window` si ce n'est l'apparition du paramètre `label`. Ce dernier permet simplement de spécifier le texte qui sera affiché par le widget crée.

Le texte du widget peut être modifié à l'aide de la méthode héritée de la classe `wx.Control`.

```
SetLabel(text)
```

De manière symétrique, le texte peut être obtenu à l'aide de la méthode `GetLabel()`.

Il est possible d'empêcher le widget de prendre trop de largeur en demandant d'insérer des retours à la ligne automatiquement après n pixel à l'aide de la méthode suivante

```
Wrap(width)
```

Attention tout de même, la coupure ne s'effectue qu'à la fin des mots. Si la taille autorisée est plus petite que certains mot, cela peut poser des problèmes.

6.2. wx.TextCtrl

A la différence de beaucoup de librairie destinée à faire des interfaces utilisateur, l'entrée de texte uniligne ou multiligne est représentée par le même widget dans wxPython.

Le constructeur se présente sous la forme suivante

```
wx.TextCtrl( parent, id=-1, value=EmptyString, pos=DefaultPosition,
             size=DefaultSize, style=0, validator=DefaultValidator,
             name=StaticTextNameStr)
```

Le paramètre `label` de `wx.StaticText` est devenu `value` ici, mais représente la même fonctionnalité.

Le paramètre `validator` permet d'associer une instance d'une classe dérivant de `wx.PyValidator`. Un validator est une classe qui permet de vérifier facilement le contenu d'un contrôle en surchargeant la méthode `Validate()` mise à disposition. Nous ne détaillerons pas cette possibilité dans le cadre de ce document.

Par défaut, un `wx.TextCtrl` ne peut avoir qu'une seule ligne de texte, il est possible de changer cela en utilisant le style `wx.TE_MULTILINE`.

Pour obtenir le texte entré par un utilisateur, plusieurs méthodes sont à disposition (toutes renvoient un `String`).

```
    GetLineText(self, lineNo)
    GetString(self, from, to)
    GetRange(self, from, to)
    GetStringSelection(self)
```

La première permet d'obtenir le texte d'une ligne donnée par son numéro. Comme dans le programme exemple, le `wx.TextCtrl` est uniligne, cet appel permet d'obtenir d'un coup tout le texte qu'il contient.

```
    self.txtEntry.GetLineText(0)
```

La seconde et la troisième méthode permet d'obtenir le texte entre deux positions de caractères en sachant que le premier caractère est à la position 0. Dans le cas d'un `wx.TextCtrl` multiligne, une manière simple d'obtenir l'entier du texte qu'il contient serait

```
    self.txtMultiligne.GetString(0, self.txtMultiligne.GetLastPosition())
```

Finalement, la quatrième méthode permet d'obtenir le texte sélectionné par l'utilisateur au sein du widget.

Il est aussi possible d'écrire du texte de manière programmée grâces aux méthodes

```
    SetValue(self, value)
    LoadFile(self, file)
    AppendText(self, text)
    WriteText(self, text)
```

La première remplace définitivement l'ancien texte contenu dans le widget par la chaîne `value`.

La seconde a le même comportement que la première, juste qu'il s'agit de mettre en place directement le contenu d'un fichier texte.

Les deux suivantes ajoutent le texte respectivement à la fin du texte existant dans le widget ou derrière le curseur d'insertion.

Dans tous les cas, à la fin de l'exécution d'une méthode d'ajout de texte, le curseur d'insertion se retrouve à la fin du texte inséré.

Au besoin, il est donc possible de manipuler le curseur d'insertion à l'aide des méthodes `GetInsertionPoint() et SetInsertionPoint()` dont vous devinerez facilement l'usage.

Finalement, sachez qu'il est possible de paramétrer complètement le style du texte de manière globale ou localisée au sein du `wx.TextCtrl`. Des méthodes sont à disposition ainsi qu'une classe `wx.TextAttr` pour cela. Nous vous laissons le soin de découvrir les possibilités offertes au sein de la documentation de référence wxPython et wxWidgets.

6.3. wx.Button

Le bouton est le principal moyen pour l'utilisateur de valider ses requêtes à l'application en plus des menus et touches mnémoniques associées.
En plus de la classe `wx.Button` permettant de créer des boutons standards contenant du texte,

wxPython met à disposition une sous classe `wx.BitmapButton` permettant de remplacer le texte par une image dans le bouton.

Pour des applications relativement évoluée, les boutons sont généralement regroupés au seins de barre d'outils pour lesquels vous trouverez des informations en regardant la classe `wx.ToolBarBase` et ses dérivées dans les manuels de références.

Le constructeur de la classe `wx.Button` se présente comme suit

```
wx.Button(parent, id=-1, label=EmptyString, pos=DefaultPosition,
          size=DefaultSize, style=0, validator=DefaultValidator,
          name=ButtonNameStr)
```

La classe ne met pas beaucoup de méthodes à disposition. Etant donné qu'elle dérive de `wx.Control`, les méthodes `SetLabel() et GetLabel()` vue pour le `wx.StaticText`, permettent de manipuler le texte contenu dans le bouton.

La gestion d'événement, propre aux boutons ayant été vue dans le chapitre 4.3, il n'y a pas lieu d'en dire plus ici.

6.4. wx.ListBox

Cette classe représente une liste de chaîne de caractères qui peuvent être sélectionnée. En général, de telles listes sont utilisée pour permettre à l'utilisateur d'effectuer plusieurs choix au sein de l'application. Dans le programme exemple, elle est utilisée comme un conteneur de données qu'il serait facile de synchroniser, par exemple, avec une petite base de données.

Il faut d'abord savoir que la `wx.ListBox`, dérive d'une interface `wx.ItemContainer` qui met à disposition les méthodes de manipulation des éléments qu'on retrouverait, notamment, dans des `wx.ComboBox`. Vous êtes fortement invité à étudier le fonctionnement de ce dernier que nous ne traiterons pas au sein de ce document.

Le constructeur se présente sous la forme suivante.

```
wx.ListBox(parent, id=-1, pos=DefaultPosition, size=DefaultSize, choices,
           style=0, validator=DefaultValidator, name=ListBoxNameStr)
```

Ce constructeur change des habituels par l'apparition du paramètre `choice`. Ce paramètre est une liste ou un tuple de chaînes de caractères qui seront lue une à une et mise dans la wx.ListBox.

Il faut aussi prendre en compte le paramètre `style`. A l'instar d'un `wx.TextCtrl`, un `wx.ListBox` peut fonctionner de deux manières. Soit n'accepter d'avoir qu'un seul élément sélectionné (`wx.LB_SINGLE`) à la fois, soit d'accepter plusieurs éléments sélectionné simultanément.

De plus, le second mode peut se comporter lui encore de deux manières. Soit un clic sur un élément le sélectionne ou le déselectionne (`wx.LB_MULTIPLE`), soit l'utilisateur peut utiliser les touches <shift> et <ctrl> pour sélectionner/déselectionner plusieurs éléments (`wx.LB_EXTENDED`).

Par défaut, le style d'un `wx.ListBox` est `wx.LB_SINGLE`.

Une `wx.ListBox` contient plein de méthode similaires à celle que l'on retrouve dans une liste Python pour ajouter, modifier ou supprimer des entrées basée sur leur index. Néanmoins, ce widget n'est pas forcément fait pour cet usage et il dispose de la méthode suivante pour recréer son contenu.

```
    Set( items )
```

Cette méthode modifie le paramètre `choice` vu dans le constructeur. Cette opération détruit les anciennes entrées de la liste.

L'intérêt d'un `wx.ListBox` étant d'offrir des choix à l'utilisateur, il faut donc pouvoir connaître les choix effectués.

Dans le cas d'un `wx.ListBox` avec un seul choix simultané possible (tel que celui du programme exemple), les méthodes suivantes permettent de connaître l'index ou, respectivement, le contenu de l'objet sélectionné.

```
    GetSelection()
    GetStringSelection()
```

Si aucun élément n'est sélectionné, la première méthode renverra une constante `wx.NOT_FOUND`, alors que la seconde renverra simplement une chaîne vide. Vous remarquerez que ce sont les mêmes méthodes que celles vues dans le chapitre parlant des boites de dialogues à choix multiple.

Dans le cas d'un `wx.ListBox` à sélection multiple, il n'y a qu'une seule méthode à disposition.

```
    GetSelections()
```

Elle renvoie un tuple des index des éléments sélectionné dans la liste.

Finalement, un `wx.ListBox` met à disposition deux événements très utiles que sont `wx.EVT_LISTBOX` et `wx.EVT_LISTBOX_DCLICK`. Le premier est généré lorsque l'utilisateur sélectionne un élément et le second lorsque l'utilisateur effectue un double-clic sur un élément de la liste.

Dans le programme exemple, les boutons "Modifier" et "Supprimer" ne s'activent que lorsque l'utilisateur a sélectionné un élément de la liste.

```
    self.Bind(wx.EVT_LISTBOX, self.OnListSelect, self.lstListe)
```

Un bon exercice pourrait être d'améliorer le programme en proposant immédiatement à l'utilisateur de modifier l'élément sur lequel il effectue un double-clic.

6.5. wx.RadioBox/wx.CheckListBox

Ces deux contrôle sont relativement sembable, ils permettent à l'utilisateur de faire un choix unique parmis plusieurs pour le `wx.RadioBox`, ou plusieurs choix simultané pour le `wx.CheckListBox`.

Le programme exemple radio_check.py vous donne un exemple de leur apparence au sein d'une interface graphique.

Les choix d'un `wx.RadioBox` sont appelés "bouton radio", une classe permettant d'en définir manuellement existe, `wx.RadioButton`. Le `wx.RadioBox` offre simplement une manière simple et commode de les grouper.

Le constructeur d'un `wx.RadioBox` est le suivant

```
wx.RadioBox(parent, id=-1, label=EmptyString, pos=DefaultPosition,
            size=DefaultSize, choices=wxPyEmptyStringArray, majorDimension=0,
            style=RA_SPECIFY_COLS, validator=DefaultValidator,
            name=RadioBoxNameStr)
```

Le paramètre `label` est un `String` représentant le texte associé au groupe de boutons radios. Dans le programme exemple, vous pouvez voir qu'il s'affiche dans la bordure.

Le paramètre `choices` est une liste ou un tuple de String qui représentent les différents choix à disposition pour l'utilisateur.

Le paramètre `majorDimension` détermine le nombre maximum de ligne ou de colonnes que le widget pourra avoir dépendant du choix du style.

Le style peut prendre la valeur `wx.RA_SPECIFY_COLS` ou `wx.RA_SPECIFY_ROWS` en plus des valeurs héritée de `wx.Window`. Ces valeurs permettent de déterminer si le paramètre `majorDimension` s'applique en colonne ou en ligne lorsque le widget disposera ses choix.

Dans le programme exemple, le paramètre a été mis à `wx.RA_SPECIFY_ROWS` et `majorDimension` à 0 (valeur neutre), donc les 4 choix s'affichent sur une ligne différente. Si nous avions mis la valeur `majorDimension` à 2 par exemple, les choix se seraient affiché sur deux colonnes, car le widget aurait été autorisé à ne contenir que 2 lignes.

A la différence d'un `wx.ListBox` (vu à la section précédente), le `wx.RadioBox` ne met aucune méthode à disposition pour changer les choix à disposition. C'est un structure purement statique au sein de l'interface utilisateur.

Il met a disposition les même méthodes qu'une `wx.ListBox` pour connaître la selection de l'utilisateur.

```
GetSelection(self)
GetStringSelection(self)
```

Ces méthodes fonctionnent comme celles d'un `wx.ListBox` et toujours comme un `wx.ListBox`, le `wx.RadioBox` met à disposition l'événement `wx.EVT_RADIOBOX` lorsque l'utilisateur click sur un bouton radio.

Le `wx.CheckListBox` est encore plus proche d'un `wx.ListBox`, puisque il en dérive directement !

De ce fait, son apparence à l'ecran diffère relativement de ce qu'on a l'habitude de voir dans les interfaces graphiques usuelles pour les cases à choix multiples.

Le constructeur d'un `wx.CheckListBox` est le suivant

```
wx.CheckListBox(parent, id, pos, size, choices, style, validator, name)
```

C'est en fait exactement le même que celui d'un `wx.ListBox`. Le `wx.CheckListBox` met simplement deux méthodes et un événement supplémentaires à disposition.

```
Check(self, item, check=true)
IsChecked(self, item)
```

La première permet de cocher (ou de décocher en passant la valeur false au paramètre check) une case par son index (paramètre item).

La seconde permet de savoir si une case est cochée ou pas à partir de son index aussi.

Il est essentiel de bien faire la différence entre un élément sélectionné et coché. La méthode héritée `GetSelections()` ne retournera pas une liste des index des cases cochée, mais bel et bien une liste des index des éléments sélectionné de manière standard dans la liste affichée.

Ainsi, pour connaître les choix de l'utilisateur, vous êtes obligé d'itérer à travers tous les index et de contrôler à chaque itération si le contrôle est coché avant de faire une action.

```
    for id in range(0, 4):
        if self.checkListBox.IsChecked(id):
            resultNumeric += int(self.checkListBox.GetString(id))
```

Pour faire cette différence entre sélection/déselection et cochage/décochage, le `wx.CheckListBox` met à disposition un nouvel événement `wx.EVT_CHECKLISTBOX`.

Vous avez sans doute remarqué que ce widget n'est pas vraiment conventionnel. L'utilisateur est généralement habitué à voir des zone composé de case à cocher ayant une apparence similaire à un `wx.RadioBox`. Un tel résultat est obtenable en assemblant un `wx.Panel` et des `wx.CheckBox` au sein d'un `wx.StaticBoxSizer` couplé à un `wx.StaticBox`. Un bon exercice serait de développer une classe `wx.CheckBoxBox`, sembable à un `wx.RadioBox` en suivant cette voie.

7. Manipuler des éléments graphiques

wxPython utilise des couches d'abstractions pour manipuler des éléments graphiques. La couche d'abstraction la plus primaire est appelée Device Context. Il s'agit d'une classe qui offre des méthodes pour dessiner sur des périphériques comme l'écran, une imprimante ou autre.

7.1. Comment charger une image

Les manipulations d'image avec wxPython font appel à deux classes, wx.Image et wx.Bitmap. La première est indépendante de la plate<forme tandis que la seconde l'est. En pratique cela implique que les manipulations se font a l'aide de wx.Image et que wx.Bitmap s'occupe de d'afficher les images à l'écran.

Il est possible de charger une image en appelant le constructeur de wx.Image

```
wx.Image (name, type=wx.BITMAP_TYPE_ANY, index=-1)
```

Le paramètre `name` est le nom de fichier de l'image à charger, le paramètre `type` est le type de l'image (cf table ci-dessous) et si le fichier contient plusieurs images (cela peu être le cas pour les fichiers GIF par exemple), le paramètre `index` permet de spécifier l'image.

Voici une liste non-exhaustive des principaux formats d'images supportés

Type (flag)	Description
`wx.BITMAP_TYPE_JPEG`	Image de type JPEG
`wx.BITMAP_TYPE_GIF`	Image de type GIF
`wx.BITMAP_TYPE_PNG`	Image de type PNG
`wx.BITMAP_TYPE_ICO`	Format des icônes Windows
`wx.BITMAP_TYPE_TIF`	Image de type TIFF
`wx.BITMAP_TYPE_ANY`	wxPython tentera de détecter automatique le type de l'image.

7.2. Manipulation d'images

La classe `wx.Image` propose différentes opérations sur les images. Vous trouverez dans la table ci-dessous les opérations les plus courantes.

Méthodes	Description
`Rescale(width, height)`	Change la taille de l'image avec les nouvelles valeurs `width` et `height`.
`Rotate(angle, rotationCentre, interpolating=True, offsetAfterRotation=None)`	Retourne un nouvel objet `wx.Image` après avoir effectué une rotation angle radians par rapport au centre `rotationCentre` (un `wx.Point`). Si `interpolating` est vrai, un algorithme plus long mais plus précis et utilisé.
`Rotate90(clockwise=True)`	Effectue une rotation de 90 degrés sur l'image. Si le paramètre est vrai, la rotation se fait dans le sens des aiguilles d'une montre.
`Scale(width, height)`	Retourne une copie de l'image après avoir redimensionné l'image aux dimensions `width` et `height`.
`Mirror(horizontally=True)`	Retourne un objet `wx.Image` ou l'image est inversée selon l'axe horizontal (si le paramètre est vrai) ou l'axe vertical (si le paramètre est faux). L'image originale n'est pas modifiée.
`Replace(r1, g1, b1, r2, g2, b2)`	Remplace chaque pixel qui possède la couleur (r1, g1, b1) par la couleur (r2, g2, b2).

Sachez qu'il est possible de jouer avec la transparence des images en leur appliquant un masque (méthode `SetMaskColor`) ou de jouer avec la valeur alpha des pixels
Nous n'aborderons pas ce sujet ici, mais vous trouveriez de plus amples informations sur le wiki de wxpython (http://wiki.wxpython.org/index.cgi/WorkingWithImages) ainsi que dans la documentation de l'API (ttp://www.wxpython.org/docs/api/frames.html).

8. Menus

wxPython gère les menus avec trois classes :
- `wx.MenuBar` s'occupe de la gestion de la barre de menu en soi ;
- `wx.Menu` pour la gestion des menus individuels (ex. menu Fichier) ou des sous-menus ;
- `wx.MenuItem` qui représente un élément final du menu

La création de menu peut se résumer en sept phases :
1. Création de la barre des menus ;
2. Attacher la barre des menus à une frame ;
3. Création des menus individuels ;
4. Attacher ce menu a la barre des menus ou a un menu parent ;
5. Création des éléments (entrées) de menus ;
6. Attacher les éléments de menus au menu approprié ;
7. Lier une action pour chaque élément de menus.

L'ordre d'exécution des actions ci-dessus est flexible et peut changer. Toutefois lors d'une création de menu, vous passerez automatiquement par les 7 étapes ci-dessus.

8.1. Création de la barre de menu

la création d'une barre de menu se fait à l'aide du constructeur `wx.MenuBar` qui n'a pas de paramètre.

Pour attacher celle-ci à une frame, il faut faire appel à la méthode `SetMenuBar(menubar)` de `wx.Frame`.

```
menubar = wx.MenuBar()
self.SetMenuBar(menubar)
```

8.2. Création des menus

le constructeur `wx.Menu()` permet de créer les menus déroulants.

```
wx.Menu(title="", style=0)
```

Il n'y a qu'un `style` possible pour les menus et il n'est supporté que par GTK. Il s'agit du style `wx.MENU_TEAROFF` qui permet de détacher le menu de sa barre afin de l'utiliser de manière autonome. Le paramètre `title` reste un mystère, nous ne l'utiliserons pas dans nos exemples.

8.3. Attacher et supprimer un menu à la bar des menus

wx.MenuBar dispose de trois méthodes intéressantes pour gérer les menus de la barre de menu.

Méthodes	Description
`Append(menu, title)`	Ajoute le menu passé en paramètre à la bar des menus en lui associant title comme nom.
`Insert(pos, menu, title)`	Insère le menu passé en paramètre dans la bar des menus à la position pos.
`Remove(pos)`	Supprime le menu situé a la postion pos.

8.4. Ajouter des entrées au menu

Il existe différentes approches pour ajouter des entrées de menu (par exemple ajouter l'entrée 'Quitter' au menu Fichier).
La méthode la plus succincte est d'utiliser la méthode `Append()` de `wx.Menu` :

```
Append ( id, text, helpStr="", kind=wx.ITEM_NORMAL )
```

Le paramètre id est un ID wxPython. Le paramètre `text` est la chaîne de caractère qui sera affichée dans le menu (le nom de l'entrée). Si l'argument `helpStr` est défini et que la frame contient une bar de status, la chaîne `helpStr` sera affichée dans la bar de status lorsque l'entrée sera mise en évidence. Cette méthode ajoute l'entrée à la fin du menu en créant une instance implicite de `wx.MenuItem`.

Il est possible d'ajouter des séparateurs au menu via la méthode `AppendSeparator()` de `wx.Menu`.

Exemple d'utilisation:

```
mnuFichier  = wx.Menu()
itemNouveau = mnuFichier.Append (wx.ID_ANY, "Nouveau")
mnuFichier.AppendSeparator()
itemQuitter = mnuFichier.Append (wx.ID_ANY, "&Quitter")
# notez le '&' pour créer un accès mnémonique sur le 'Q'
```

Une deuxième approche pour ajouter des entrées à un menu est de créer explicitement les entrées (via le constructeur `wx.MenuItem`) puis de les ajouter au menu. Voici le constructeur permettant de créer les entrées de menu :

```
wx.MenuItem(parentMenu=None, id=wx.ID_ANY, text="", helpStr="",
            kind=wx.ITEM_NORMAL, subMenu=None)
```

Le paramètre `parentMenu` représente le menu parent de l'item et doit être une instance de `wx.Menu`. Le fait de préciser le parent ici ne va pas automatiquement ajouter l'entrée au menu! Il faut le faire explicitement avec la méthode `AppendItem()`.
Vous avez la possibilité de transformer cette entrée en sous-menu en précisant le paramètre `subMenu`. Toutefois nous vous conseillons de ne pas utiliser cette méthode, mais d'utiliser l'approche présentée dans la section "Sous-Menu" de ce chapitre.

Voici une autre façon de faire pour avoir le même résultat:

```
mnuFichier  = wx.Menu()
itemNouveau = wx.MenuItem(mnuFichier, wx.ID_ANY, "Nouveau")
itemQuitter = wx.MenuItem(mnuFichier, wx.ID_ANY, "&Quitter")

mnuFichier.AppendItem(itemNouveau)
mnuFichier.AppendSeparator()
mnuFichier.AppendItem(itemQuitter)
```

8.5. Gestion d'évènement de menu

Lier une action à une entrée de menu revient au même que de lier une action à un bouton (à l'évènement près). Le principe reste le même, il faut utiliser la méthode `Bind()` sur la frame contenant la bar de menu avec l'évènement `wx.EVT_MENU`.

```
self.Bind(wx.EVT_MENU, self.OnQuitter, itemQuitter)
```

Où `self` est la frame, `self.OnExit` la méthode à appeler et `itemQuitter` l'entrée de menu.

8.6. Raccourcis clavier et accès mnémoniques

Il est possible de rajouter des touches mnémoniques aux entrées de menu en rajouter un «et commerciale» (&) avant la lettre cible du texte à afficher. Par exemple pour lier un accès mnémonique sur le 'n' de `"Enregistrer sous..."` il faut écrire `"E&nregistrer sous..."`.

De même, il existe un moyen simple pour rajouter des raccourcis claviers à un item d'un menu. Il suffit d'inclure la combinaison de touche dans le texte à afficher de l'élément de menu. Pour appliquer cela, ajoutez le caractère de tabulation `\t` à la fin du texte suivi de la combinaison de touches. Cette combinaison peut commencer par l'une de ses touches de modification `Ctrl`, `Alt` ou `Shift` puis le séparateur `'+'` ou `'-'` (comportement identique) suivi du caractère voulu.

Un exemple sera plus clair : vous souhaitez attribuer le raccourcis clavier `Ctrl+S` à l'entrée `"Enregistrer"`, il vous suffit donc d'écrire `"Enregistrer\tCtrl+S"` (ou `"Enregistrer\tCtrl-S"`). Il est possible de combiner les touches de modifications par exemple `"Enregistrer Sous\tCtrl+Shit+S"`. Magique, non ?

Sous MacOS X, la touche de modification `Ctrl` représente la touche Command (Pomme).

```
itemSauveSous = mnuFichier.Append(wx.ID_ANY, "E&nregistrer Sous...\tCtrl-Shit-S")
```

8.7. Sous-menu

Pour ajouter des sous-menu la classe `wx.Menu` offre une méthode `AppendMenu()` qui a la comme signature

```
AppendMenu ( id, text, submenu )
```

Prenons directement un exemple

```
    menubar = wx.MenuBar()
    menu = wx.Menu()
    sousmenu = wx.Menu()
    item1 = sousmenu.Append(wx.ID_ANY, "Element sous-menu 1")
    item2 = sousmenu.Append(wx.ID_ANY, "Element sous-menu 2")

    # ajout du sous menu au menu
    menu.AppendMenu(wx.ID_ANY, "Sous-menu", sousmenu)
    itemQuitter = menu.Append(wx.ID_ANY, "Quitter")

    menubar.Append (menu, "Menu")
    self.SetMenuBar(menubar)
```

9. Conclusion

Au terme de ce document, nous espérons que vous avez eu la possibilité de vous forger une vue d'ensemble de wxPython et de ses fonctionnalités.

Nous sommes conscient que tout n'a pas été couvert et que l'absence de documentation en français reste un problème pour cette librairie. Néanmoins, la bibliographie anglaise, et notamment l'excellent livre, "wxPython in action"[1] reste parfaitement abordable pour les gens ne maîtrisant pas complétement la langue de Shakespeare.

De plus, la communauté wxPython est très active et le wiki propose de nombreux exemples complets et bien expliqués. Elle développe aussi des outils, comme des concepteurs d'interfaces, qui sont en phase d'être finalisés. Bien que nous n'en parlions pas dans ce document, nous vous recommandons de jeter un oeil aux applications wxGlade et BoaConstructor.

10. Bibliographie

[1] wxPython in action, Noel Rappin et Robin Dunn, editions Manning 2006
[2] Site wxPython.org, http://www.wxpython.org
[3] Site wxWidgets.org, http://www.wxwidgets.org
[4] Wiki wxPython, http://wiki.wxpython.org
[5] Site MacPython, "Jack", http://homepages.cwi.nl/~jack/macpython

11. Code sources des exemples

11.1. Hello World (hello.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

class MainApp(wx.App):
    """ Création de l'application qui dérive elle de la classe application
        offerte par wxPython.
        Attention: Le "constructeur" est un peu spécial !"""
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('Hello World !')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Création d'une fenetre adaptée à notre application. La classe fenêtre
        de base offerte par wxPython est dérivée à cet effet. """
    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title)
        self.Show(True)

if __name__ == "__main__":
    app = MainApp()
    app.MainLoop()
```

11.2. Exemple de gestion d'événements simple (goodbye.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

class MainApp(wx.App):
    """ Création de l'application qui dérive elle de la classe application
        offerte par wxPython.
        Attention: Le "constructeur" est un peu spécial !"""
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('Hello World !')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Création d'une fenetre adaptée à notre application. La classe fenêtre
        de base offerte par wxPython est dérivée à cet effet. """
    def __init__(self, title):
        """
            Changement ici, spécification de la taille de la frame ainsi
            qu'une expression avancée pour le style
            Cette expression dit d'utiliser le style par défaut des fenêtres
            et de retirer (opérateur ^) la possibilité de redimensionner la
            fenêtre ainsi que les trois boutons de la barre de titre
        """
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(300, 100),
                          style=wx.DEFAULT_FRAME_STYLE ^ ( wx.RESIZE_BORDER |
                                                           wx.MINIMIZE_BOX  |
                                                           wx.MAXIMIZE_BOX  |
                                                           wx.CLOSE_BOX) )
        # Ajout d'un conteneur de widget dans la frame
        panel = wx.Panel(self, wx.ID_ANY)
        # Ajout d'un bouton dans le conteneur a une position choisie
        self.button = wx.Button(panel, wx.ID_ANY, "Good bye !", pos=(100, 30))
        # Affectation de la methode OnClick a l'evenement EVT_BUTTON pour
        # l'objet button
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
        self.Show(True)

    def OnClick(self, event):
        #wx.Exit()
        self.Close()

if __name__ == "__main__":
    app = MainApp()
    app.MainLoop()
```

11.3. Boîtes de dialogues (dialog.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

class MainApp(wx.App):
    """ Création de l'application qui dérive elle de la classe application
        offerte par wxPython."""
    def OnInit(self):

        # Création de la fenêtre principale
        frame = MainWindow('Boite de dialogue')
        self.SetTopWindow(frame)
        return True

    def OnExit(self):
        """ Methode appelée automatiquement avant que l'application quitte"""

        # instanciation d'une boite de dialogue simple
        dialogue = wx.MessageDialog(None, 'GoodBye!', "message d'au revoir",
                                    wx.OK)
        # affichage de celle-ci
        result = dialogue.ShowModal()
        return True

class MainWindow(wx.Frame):
    """ Création d'une fenetre adaptée à notre application. La classe fenêtre
        de base offerte par wxPython est dérivée à cet effet. """

    def __init__(self, title):
        # Changement ici, spécification de la taille de la frame
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(300, 150))

        # Ajout d'un conteneur de widget dans la frame
        self.panel = wx.Panel(self, wx.ID_ANY)

        # Sizer pour placer les boutons
        sizer = wx.GridBagSizer(5,5)


        # Ajout des boutons dans le conteneur
        btnDialog = wx.Button(self.panel, wx.ID_ANY, "Boite de dialogue simple",
                              size=(200, 15))
        btnSaisie = wx.Button(self.panel, wx.ID_ANY, "Boite de dialogue de saisie",
                              size=(200, 15))
        btnChoix1  = wx.Button(self.panel, wx.ID_ANY, "Boite a choix unique",
                               size=(200, 15))
        btnChoix2  = wx.Button(self.panel, wx.ID_ANY, "Boite a choix multiple",
                               size=(200, 15))
        btnChoix3  = wx.Button(self.panel, wx.ID_ANY, "Choix d'une couleur",
                               size=(200, 15))
        btnExit    = wx.Button(self.panel, wx.ID_ANY, "&Quitter")

        # Ajout au sizer
        sizer.Add(btnDialog, (0,0), flag=wx.ALIGN_CENTER|wx.EXPAND|wx.ALL, border=5)
        sizer.Add(btnSaisie, (1,0), flag=wx.ALIGN_CENTER|wx.EXPAND|wx.ALL, border=5)
        sizer.Add(btnChoix1, (2,0), flag=wx.ALIGN_CENTER|wx.EXPAND|wx.ALL, border=5)
        sizer.Add(btnChoix2, (3,0), flag=wx.ALIGN_CENTER|wx.EXPAND|wx.ALL, border=5)
        sizer.Add(btnChoix3, (4,0), flag=wx.ALIGN_CENTER|wx.EXPAND|wx.ALL, border=5)
        sizer.Add(btnExit,   (5,1), flag=wx.ALIGN_BOTTOM|wx.ALIGN_RIGHT)
```

```
        sizer.Add((10,10), pos=(6,0), span=(1, 3))  # marge inférieur
        sizer.Add((10,10), pos=(0,2), span=(5, 1))  # marge de droite

        sizer.AddGrowableRow(5)
        sizer.AddGrowableCol(0)
        sizer.AddGrowableCol(1)

        self.panel.SetSizer(sizer)
        sizer.Fit(self)

        # Affectation des methodes lors d'un clic sur le bouton
        self.Bind(wx.EVT_BUTTON, self.AfficherDlgSimple, btnDialog)
        self.Bind(wx.EVT_BUTTON, self.AfficherDlgSaisie, btnSaisie)
        self.Bind(wx.EVT_BUTTON, self.AfficherDlgChoixUnique, btnChoix1)
        self.Bind(wx.EVT_BUTTON, self.AfficherDlgChoixMulti,  btnChoix2)
        self.Bind(wx.EVT_BUTTON, self.AfficherDlgCouleur,  btnChoix3)

        self.Bind(wx.EVT_BUTTON, self.Quitter, btnExit)

        self.SetMinSize((300, 150))
        self.Show(True)

    def Quitter(self, source):
        self.Close()


    def AfficherDlgSimple(self, source):
        dlgMsg = wx.MessageDialog(None, 'Aimez-vous cet exemple ?',
                                  'Question',
                                  wx.YES_NO | wx.YES_DEFAULT | wx.ICON_QUESTION)
        reponse = dlgMsg.ShowModal()
        if reponse == wx.ID_YES:
            message = 'Chouette!'
        else:
            message = 'Oh, non!'
        wx.MessageDialog(self, message, style=wx.OK).ShowModal()
        dlgMsg.Destroy()

    def AfficherDlgSaisie(self, source):
        dlgSaisie = wx.TextEntryDialog(self, 'Entrez votre prenom',
                                       'Question personnelle',
                                       'entrer le texte ici',
                                        style=wx.OK|wx.CANCEL)
        # Affichage de la boite de dialogue + recuperation de la valeur de retour
        reponse = dlgSaisie.ShowModal()
        if reponse == wx.ID_OK:
            prenom = dlgSaisie.GetValue()
            wx.MessageDialog(None, 'Bonjour ' + prenom, style=wx.OK).ShowModal()
        dlgSaisie.Destroy()


    def AfficherDlgChoixUnique(self, source):
        dlgMulti = wx.SingleChoiceDialog(self,
                    'Quel est votre fruit prefere ?',
                    'Choix unique',
                    ['peche', 'pomme', 'abricot', 'prune', 'poire'])
        if dlgMulti.ShowModal() == wx.ID_OK:
            fruit = dlgMulti.GetStringSelection()
            wx.MessageDialog(None,
                    "Prepartion d'une tarte aux " + fruit + "s en cours...",
                    style=wx.OK).ShowModal()
        dlgMulti.Destroy()
```

```
    def AfficherDlgChoixMulti(self, source):
        fruits = ['peche', 'pomme', 'abricot', 'prune', 'poire']

        dlgMulti = wx.MultiChoiceDialog(self,
                    'Quels sont vos fruits preferes ?',
                    'Choix multiple', fruits)

        if dlgMulti.ShowModal() == wx.ID_OK:
            # recuperation des elements selectionne sous la forme d'une liste d'index
            listeFruits = ""
            for i in dlgMulti.GetSelections():
                listeFruits = listeFruits + fruits[i] + ", "
            wx.MessageDialog(None,
                    "Prepartion d'une tarte aux " + listeFruits + " en cours...",
                    style=wx.OK).ShowModal()
        dlgMulti.Destroy()


    def AfficherDlgCouleur(self, source):
        dlgCouleur = wx.ColourDialog(self)
        res = dlgCouleur.ShowModal()
        if res == wx.ID_OK:
            # recuperation du DataColour
            couleur = dlgCouleur.GetColourData()
            # colorise le fond du panel avec la couleur choisie
            self.panel.SetBackgroundColour(couleur.GetColour())
        dlgCouleur.Destroy()



if __name__ == "__main__":
    app = MainApp(redirect=False)
    app.MainLoop()
```

11.4. Exemple d'utilisation du GridSizer (gridsizer.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

# liste de noms des boutons pour l'exemple
labels = "un deux trois quatre cinq six sept huit neuf".split()

class MainApp(wx.App):
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('GridSizer')
        self.SetTopWindow(frame)
        return True


class MainWindow(wx.Frame):
    """Fenetre pour presenter le Grid sizer. """

    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title=title, size=(300, 150))

        # creation d'un panel
        self.panel = wx.Panel(self);

        # creation du grid sizer
        self.sizer = wx.GridSizer(rows=3, cols=3, hgap=2, vgap=2)

        # creation et ajout des boutons au sizer
        for label in labels:
            # Dans cet exemple on affecte aucune action derriere les boutons.
            btn = wx.Button(self.panel, wx.ID_ANY, label);
            self.sizer.Add(btn, 0, 0)

        # lie le sizer au panel
        self.panel.SetSizer(self.sizer)
        # ajuste la taille du sizer
        self.sizer.Fit(self)
        # definition de la taille minimale de la fenetre
        self.SetMinSize((300, 150))

        # Affichage de la fenetre
        self.Show(True)


if __name__ == "__main__":
    app = MainApp(redirect=False) # pour ne pas rediriger le flux de sortie
                                  # standard dans la fenetre graphique
                                  # (utile pour faire du debuggage)
    app.MainLoop()
```

11.5. Exemple d'utilisation du FlexGridSizer (flexgridsizer.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

# liste de noms des boutons pour l'exemple
labels   = "un deux trois quatre cinq six sept huit neuf dix onze douze".split()
couleurs = ["red", "gray", "sky blue"]


class MainWindow(wx.Frame):
    """Fenetre pour presenter le Flex Grid sizer. """

    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title=title, size=(300, 120))

        # creation du grid sizer
        sizer = wx.FlexGridSizer(rows=4, cols=3, hgap=2, vgap=2)
        i=0
        # creation et ajout des panels au sizer
        for label in labels:
            panel = PanelColore(self, wx.ID_ANY, label, couleurs[i%len(couleurs)]);
            sizer.Add(panel, 0, wx.EXPAND) # wx.EXPAND permet d'etendre la taille du
                                           # panel
            i=i+1

        sizer.AddGrowableCol(2) # rend la 3eme colonne extensible
        sizer.AddGrowableRow(1) # rend la 2eme ligne extensible
        sizer.AddGrowableRow(3) # rend la 2eme ligne extensible

        # lie le sizer a la frame
        self.SetSizer(sizer)
        # ajuste la taille du sizer
        self.Fit()
        # definition de la taille minimale de la fenetre
        self.SetMinSize((300, 120))

        # Affichage de la fenetre
        self.Show(True)


class PanelColore (wx.Panel):
    """ Panel avec un fond de couleur pour bien distinguer les différents
        alignement des sizers.
    """
    def __init__(self, parent, ID=wx.ID_ANY, label="", color="white",
pos=wx.DefaultPosition, size=(100, 30)):

        wx.Panel.__init__(self, parent, ID, pos, size, name=label)
        self.label=label
        self.SetBackgroundColour(color);
        self.SetMinSize(size)
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, event):
        size = self.GetSize() # recuperation de la taille du panel
        dc = wx.PaintDC(self) # recuperation du contexte graphique
                              # (DC = Device Context)
        # recuperation de la largeur et hauteur que prendra le texte a afficher
        w,h = dc.GetTextExtent(self.label)
        # dessine le texte dans le panel
```

```
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)


class MainApp(wx.App):
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('FlexGridSizer')
        self.SetTopWindow(frame)
        return True


if __name__ == "__main__":
    app = MainApp(redirect=False) # pour ne pas rediriger le flux de sortie
                                  # standard dans la fenetre graphique
                                  # (utile pour faire du debuggage)
    app.MainLoop()
```

11.6. Exemple d'utilisation du GridBagSizer (gridbagsizer.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx


# liste de noms des boutons pour l'exemple
couleurs = ["red", "gray"]
# liste de noms des boutons pour l'exemple
labels   = "un deux trois quatre cinq six".split()

class MainWindow(wx.Frame):
    """Fenetre pour presenter le Grid Bag sizer. """

    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, size=(400, 120))

        # creation du grid bag sizer
        sizer = wx.GridBagSizer(hgap=2, vgap=2)

        # creation et ajout de 6 panels au sizer
        for colonne in [0, 1]:
            for ligne in range(3):
                panel = PanelColore(self, wx.ID_ANY, labels[colonne + ligne*2],
                                    couleurs[colonne]);
                sizer.Add(panel, pos=(ligne, colonne))

        # Ajout du panel qui sera positionné sur toute la 3 colonnes
        panel =  PanelColore(self, wx.ID_ANY, "fusion de 4 lignes", "sky blue");
        sizer.Add(panel, pos=(0, 2), span=(4,1), flag=wx.EXPAND)

        # Ajout du panel qui sera positionné dans les 2 premieres colonnes
        # de la derniere ligne
        panel =  PanelColore(self, wx.ID_ANY, "fusion de 2 colonnes", "gold");
        sizer.Add(panel, pos=(3, 0), span=(1,2), flag=wx.EXPAND)

        sizer.AddGrowableCol(2) # rend la 3eme colonne extensible
        sizer.AddGrowableRow(3) # rend la 4eme ligne extensible

        # lie le sizer a la frame
        self.SetSizer(sizer)

        # ajuste la taille du sizer
        sizer.Fit(self)

        # Affichage de la fenetre
        self.Show(True)


class PanelColore (wx.Panel):
    """ Panel avec un fond de couleur pour bien distinguer les différents
        alignement des sizers.
    """
    def __init__(self, parent, ID=wx.ID_ANY, label="", color="white",
                 pos=wx.DefaultPosition, size=(100, 30)):

        wx.Panel.__init__(self, parent, ID, pos, size, name=label)
        self.label=label
        self.SetBackgroundColour(color);
        self.SetMinSize(size)
        self.Bind(wx.EVT_PAINT, self.OnPaint)
```

```
    def OnPaint(self, event):
        size = self.GetSize() # recuperation de la taille du panel
        dc = wx.PaintDC(self) # recuperation du contexte graphique
                              # (DC = Device Context)
        # recuperation de la largeur et hauteur que prendra le texte a afficher
        w,h = dc.GetTextExtent(self.label)
        # dessine le texte au centre du panel
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)


class MainApp(wx.App):
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('Grid Bag Sizer')
        self.SetTopWindow(frame)
        return True


if __name__ == "__main__":
    app = MainApp(redirect=False)
    app.MainLoop()
```

11.7. Exemple d'utilisation du BoxSizer (boxsizer.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx


class MainWindow(wx.Frame):
    """Fenetre pour presenter le Box sizer. """

    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, size=(300, 150))

        # creation du grid sizer
        sizer = wx.BoxSizer(wx.VERTICAL)

        # definition des panels
        haut    = PanelColore(self, wx.ID_ANY, "Panel du haut", "red");
        milieu  = PanelColore(self, wx.ID_ANY, "Panel du milieu", "coral");
        bas     = PanelColore(self, wx.ID_ANY, "Panel du bas", "lime green");

        # ajout des boutons au sizer
        sizer.Add(haut,    proportion=0, flag=wx.EXPAND)
        """ la proportion est a 1 afin que ce widget prenne tous l'espace vide """
        sizer.Add(milieu,  proportion=1, flag=wx.EXPAND)
        sizer.Add(bas,     proportion=0, flag=wx.EXPAND)

        # lie le sizer a la frame
        self.SetSizer(sizer)

        # ajuste la taille du sizer
        self.Fit()

        # Affichage de la fenetre
        self.Show(True)


class PanelColore (wx.Panel):
    """ Panel avec un fond de couleur pour bien distinguer les différents
        alignement des sizers.
    """
    def __init__(self, parent, ID=wx.ID_ANY, label="", color="white",
                 pos=wx.DefaultPosition, size=(100, 30)):

        wx.Panel.__init__(self, parent, ID, pos, size, name=label)
        self.label=label
        self.SetBackgroundColour(color);
        self.SetMinSize(size)
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, event):
        size = self.GetSize() # recuperation de la taille du panel
        dc = wx.PaintDC(self) # recuperation du contexte graphique (DC = Device
                              # Context)
        # recuperation de la largeur et hauteur que prendra le texte a afficher
        w,h = dc.GetTextExtent(self.label)
        # dessine le texte dans le panel
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)


class MainApp(wx.App):
    def OnInit(self):
```

```
        # Création de la fenêtre principale
        frame = MainWindow('GridSizer')
        self.SetTopWindow(frame)
        return True


if __name__ == "__main__":
    app = MainApp(redirect=False) # pour ne pas rediriger le flux de sortie
                                  # standard dans la fenetre graphique
                                  # (utile pour faire du debuggage)
    app.MainLoop()
```

11.8. Exemple d'utilisation du StaticBoxSizer (staticboxsizer.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx


class MainWindow(wx.Frame):
    """Fenetre pour presenter le Static Box sizer. """

    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title=title, size=(300, 150))

        # creation d'un panel
        self.panel = wx.Panel(self)

        # creation d'un static box et du sizer qui va avec
        groupe1 = wx.StaticBox(self.panel, wx.ID_ANY, "Groupe 1")
        sizerGrp1 = wx.StaticBoxSizer(groupe1, wx.VERTICAL)

        # meme operation pour le deuxieme
        groupe2 = wx.StaticBox(self.panel, wx.ID_ANY, "Groupe 2")
        sizerGrp2 = wx.StaticBoxSizer(groupe2, wx.VERTICAL)


        # definition des boutons et ajout au 2 static box sizer
        for i in range(3):
            btn = wx.Button(self.panel, wx.ID_ANY, "Bouton %d" % (i+1))
            sizerGrp1.Add(btn, flag=wx.ALL, border=3)

        for i in range(3):
            btn = wx.Button(self.panel, wx.ID_ANY, "Action %d" % (i+1))
            sizerGrp2.Add(btn, flag=wx.ALL, border=3)


        # creation du sizer englobant pour positionner les deux static box
        sizer = wx.BoxSizer(wx.HORIZONTAL)
        sizer.Add(sizerGrp1, flag=wx.ALL, border=10)
        sizer.Add(sizerGrp2, flag=wx.ALL, border=10)

        # lie le sizer au panel
        self.panel.SetSizer(sizer)

        # ajuste la taille du sizer
        sizer.Fit(self)

        # Affichage de la fenetre
        self.Show(True)


class MainApp(wx.App):
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('Static Box Sizer')
        self.SetTopWindow(frame)
        return True


if __name__ == "__main__":
    app = MainApp(redirect=False)
    app.MainLoop()
```

11.9. Exemple d'application (gestion_liste.py)

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import wx

class ListControler:
    """ Tentative de séparer un minimum l'interface graphique des données
        et logique système dans un esprit MVC

        Note: Les conventions de nommage dans cette classe sont celle du PEP 8
        puisque elle est en majorité en dehors du contexte wxPython
    """
    def __init__(self):
        self.list = []

    def get_list(self):
        """ Permet d'obtenir une référence sur la liste interne.
        """
        return self.list

    def add_element(self, position=0, text=""):
        """ Ajoute un élément dans la liste si c'est possible
        """
        if len(self.list) >= position and position >= 0:
            self.list.insert(position, text)
        else:
            self.list.append(text)

    def del_element(self, position):
        """ Retire un élément de la liste si c'est possible
        """
        if len(self.list) >= position and position >= 0:
            del self.list[position]

class MainWindow(wx.Frame):
    """ Fenêtre principale de l'application. Il n'y en aura qu'une
        de toute manière
    """
    def __init__(self, title):
        # La frame a une taille initiale de 640*480 et ne pourra pas aller
        # en dessous
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(640, 480))
        self.SetMinSize(self.GetSize())

        # Création des divers widgets de la frame comme donnée membre
        # de cette dernière.
        self.panel = wx.Panel(self)

        self.lblEntry = wx.StaticText(self.panel, wx.ID_ANY, "Entree")

        # Le contrôle d'entrée commence naturellement avec le focus
        self.txtEntry = wx.TextCtrl(self.panel, wx.ID_ANY)
        self.txtEntry.SetFocus()

        # La liste est désactivée car elle est vide au début
        self.lstListe = wx.ListBox(self.panel, wx.ID_ANY)
        self.lstListe.Enable(False)

        # Le bouton enregistrer vient avant la liste dans la liste de focus
        # par tabulation et de plus represente l'action principale du programme
        # activée par un appui sur la touche <Enter>
```

```
        self.cmdEnregistrer = wx.Button(self.panel, wx.ID_ANY, "Enregistrer")
        self.cmdEnregistrer.MoveBeforeInTabOrder(self.lstListe)
        self.cmdEnregistrer.SetDefault()

        # Les boutons de contrôle de la liste doivent être désactivé au
        # début
        self.cmdModifier = wx.Button(self.panel, wx.ID_ANY, "Modifier")
        self.cmdModifier.Enable(False)
        self.cmdSupprimer = wx.Button(self.panel, wx.ID_ANY, "Supprimer")
        self.cmdSupprimer.Enable(False)

        self.cmdQuitter = wx.Button(self.panel, wx.ID_ANY, "Quitter")

        # Association de touches mnémoniques
        table = [(wx.ACCEL_ALT, ord('E'), self.cmdEnregistrer.GetId()),
                 (wx.ACCEL_ALT, ord('M'), self.cmdModifier.GetId()),
                 (wx.ACCEL_ALT, ord('S'), self.cmdSupprimer.GetId()),
                 (wx.ACCEL_ALT, ord('Q'), self.cmdQuitter.GetId())]

        self.SetAcceleratorTable(wx.AcceleratorTable(table))

        # Une chaîne constante pour le message d'avertissement lors de la
        # suppression
        self.confirmSupprMess = "Voulez vous vraiment supprimer l'element " \
                                + " selectionne ?"

        # Création d'une liste indépendante de l'interface
        self.listObject = ListControler()

        # Deux méthodes privées permettent de créer l'affichage de base
        # et de faire les associations événements-méthode-widget nécessaire
        self.__do_layout()
        self.__do_binding()

        self.modeModification = False

    def __do_layout(self):
        """ Créer et dispose les widgets dans la fenêtre. C'est une application
            complète de la théorie concernant les sizers.
        """
        # Ce programme est divisé en deux ligne, une pour le TextCtrl
        # d'entrée et le bouton enregistrer, l'autre pour le reste.
        mainSizer = wx.FlexGridSizer(rows=2)
        mainSizer.AddGrowableCol(0)
        mainSizer.AddGrowableRow(1)

        # Ce sizer gère les widgets d'entrée utilisateur
        #(txtEntry, cmdEnregistrer)
        entrySizer = wx.FlexGridSizer(cols=3, hgap=30)
        entrySizer.AddGrowableCol(1)
        entrySizer.Add(self.lblEntry, 0, wx.ALIGN_CENTER)
        entrySizer.Add(self.txtEntry, 0, wx.EXPAND)
        entrySizer.Add(self.cmdEnregistrer, 0, 0)

        # Celui-ci gère ce qui est inhérent à la liste ainsi que le bouton
        # quitter
        listSizer = wx.FlexGridSizer(cols=2, hgap=30)
        listSizer.AddGrowableCol(0)
        listSizer.AddGrowableRow(0)
        listSizer.Add(self.lstListe, 0, wx.EXPAND)

        # Celui-ci permet de garder l'alignement des boutons en colonne
        listButtonsSizer = wx.FlexGridSizer(rows=3, vgap=10)
```

```
        listButtonsSizer.Add(self.cmdModifier, 0, 0)
        listButtonsSizer.Add(self.cmdSupprimer, 0, 0)
        listButtonsSizer.AddGrowableRow(2)
        listButtonsSizer.Add(self.cmdQuitter, 0, wx.ALIGN_BOTTOM)

        listSizer.Add(listButtonsSizer, 0, wx.EXPAND | wx.ALL, 0)

        mainSizer.Add(entrySizer, 0, wx.EXPAND | wx.ALL, 10)
        mainSizer.Add(listSizer, 0, wx.EXPAND | wx.ALL, 10)

        self.panel.SetAutoLayout(True)
        self.panel.SetSizer(mainSizer)
        self.Layout()

    def __do_binding(self):
        """ Créer ici l'ensemble de la gestion d'événement sur cette fenêtre
        """
        self.Bind(wx.EVT_BUTTON, self.OnClickEnregistrer, self.cmdEnregistrer)
        self.Bind(wx.EVT_BUTTON, self.OnClickSupprimer, self.cmdSupprimer)
        self.Bind(wx.EVT_BUTTON, self.OnClickModifier, self.cmdModifier)
        self.Bind(wx.EVT_BUTTON, self.OnClickQuitter, self.cmdQuitter)
        self.Bind(wx.EVT_LISTBOX, self.OnListSelect, self.lstListe)

    def OnClickEnregistrer(self, event=None):
        """ Click sur le bouton "Enregistrer"
        """

        # Si le mode "Modification" est activé, il faut modifier l'élément
        # qui était sélectionné dans la liste
        if self.modeModification:
            self.listObject.del_element(self.selPos)
            self.listObject.add_element(self.selPos,
                                        self.txtEntry.GetLineText(0))
            self.modeModification = False

        # Sinon ajouter le texte en bas de la liste
        else:
            self.listObject.add_element(text=self.txtEntry.GetLineText(0))

        # Synchronization de la liste avec celle du contrôleur
        self.lstListe.Set(self.listObject.get_list())

        # L'entrée du texte est effacée, regagne le focus et la liste devient
        # active pour sélectionner des éléments à l'intérieur
        self.txtEntry.SetValue("")
        self.txtEntry.SetFocus()
        self.lstListe.Enable(True)

    def OnClickModifier(self, event=None):
        """ Click sur le bouton "Modifier
        """

        # Active le mode modification et retient l'index de la selection
        self.modeModification = True
        self.selPos = self.lstListe.GetSelection()

        # Le contrôle d'entrée prend la valeur sélectionnée et gagne le focus
        self.txtEntry.SetValue(self.lstListe.GetStringSelection())
        self.txtEntry.SetFocus()

        # La liste est ses contrôles sont désactivés pour éviter des erreurs
        # de la part de l'utilisateur
        self.lstListe.Enable(False)
```

```
        self.cmdModifier.Enable(False)
        self.cmdSupprimer.Enable(False)

    def OnClickSupprimer(self, event=None):
        """ Click sur le bouton "Supprimer"
        """

        # Affichage d'un message demandant confirmation
        confirmSuppr = wx.MessageDialog(self, self.confirmSupprMess,
                                        style=wx.YES_NO | wx.NO_DEFAULT)

        # L'élément est supprimé de la liste sur confirmation de l'utilisateur
        if  confirmSuppr.ShowModal() ==  wx.ID_YES:
            self.listObject.del_element(self.lstListe.GetSelection())
            self.lstListe.Set(self.listObject.get_list())

    def OnClickQuitter(self, event=None):
        """ Click sur le bouton "Quitter"
        """
        self.Close()

    def OnListSelect(self, event=None):
        """ Selection d'un élément dans la liste
        """
        self.cmdModifier.Enable(True)
        self.cmdSupprimer.Enable(True)

class MainApp(wx.App):
    """ Gestionnaire de l'application et des fenêtres de celle-ci
    """
    def OnInit(self):
        wx.InitAllImageHandlers()
        frame = MainWindow(title="Editeur de liste")
        self.SetTopWindow(frame)
        frame.Show()
        return 1


if __name__ == "__main__":
    app = MainApp()
    app.MainLoop()
```

11.10. Exemple de boutons radio et cochable(radio_check.py)

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import wx


class MainWindow(wx.Frame):
    """ Fenêtre principale de l'application. Il n'y en aura qu'une
        de toute manière
    """
    def __init__(self, title):
        # La frame a une taille initiale de 320*200 et ne pourra pas aller
        # en dessous
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(320, 200))
        self.SetMinSize(self.GetSize())
        self.SetMaxSize(self.GetSize())

        # Création des divers widgets de la frame comme donnée membre
        # de cette dernière.
        self.panel = wx.Panel(self)

        self.radioBox = wx.RadioBox(self.panel, wx.ID_ANY,
                                    "Frequence principale",
                                    style = wx.RA_SPECIFY_ROWS,
                                    choices = ("70", "80", "90", "100"),
                                    majorDimension = 0)

        self.checkListBox = wx.CheckListBox(self.panel, wx.ID_ANY,
                                            size = (-1, 114),
                                            style = wx.LB_NEEDED_SB,
                                            choices = ("1", "2", "3", "4"))

        self.resultat = wx.StaticText(self.panel, wx.ID_ANY, "70.0 Hz")

        # Deux méthodes privées permettent de créer l'affichage de base
        # et de faire les associations événements-méthode-widget nécessaire
        self.__do_layout()
        self.__do_binding()

    def __do_layout(self):
        """ Créer et dispose les widgets dans la fenêtre. C'est une application
            complète de la théorie concernant les sizers.
        """
        # Ce programme est divisé en deux ligne, une pour les radios
        # boutons et checkboxes, l'autre pour le résultat
        mainSizer = wx.FlexGridSizer(rows=2)

        # Ce sizer gère les widgets d'entrée utilisateur
        #(radioBox, checkListBox)
        entrySizer = wx.FlexGridSizer(cols=2, hgap=30)
        entrySizer.Add(self.radioBox, 0, 0)
        entrySizer.Add(self.checkListBox, 0, 0)

        mainSizer.Add(entrySizer, 0, 0)
        mainSizer.Add(self.resultat, 0, wx.ALIGN_CENTER)

        self.panel.SetAutoLayout(True)
        self.panel.SetSizer(mainSizer)
        self.Layout()

    def __do_binding(self):
```

```
        """ Créer ici l'ensemble de la gestion d'événement sur cette fenêtre
        """
        self.Bind(wx.EVT_RADIOBOX, self.ChangeResult, self.radioBox)
        self.Bind(wx.EVT_CHECKLISTBOX, self.ChangeResult, self.checkListBox)

    def ChangeResult(self, event):
        """ Change le résultat qui est la somme des éléments sélectionnés
        """
        resultNumeric = int(self.radioBox.GetStringSelection())

        for id in range(0, 4):
            if self.checkListBox.IsChecked(id):
                resultNumeric += int(self.checkListBox.GetString(id))

        self.resultat.SetLabel(str(resultNumeric) + ".0 Hz")

class MainApp(wx.App):
    """ Gestionnaire de l'application et des fenêtres de celle-ci
    """
    def OnInit(self):
        wx.InitAllImageHandlers()
        frame = MainWindow(title="Generateur de frequence")
        self.SetTopWindow(frame)
        frame.Show()
        return 1


if __name__ == "__main__":
    app = MainApp()
    app.MainLoop()
```

11.11. Exemple d'utilisation des images (images.py)

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

class Fenetre(wx.Frame):
    """ Fenetre dans laquelle on présente le dessin et le chargement des images
simple """
    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(100, 300))
        panel = wx.Panel(self)
        sizer = wx.GridBagSizer(5,5)

        # Chargements des images
        imgPommes = wx.Image("pomme.jpg", wx.BITMAP_TYPE_JPEG)
        imgAbricots = wx.Image("abricot.png", wx.BITMAP_TYPE_ANY)

        imgPommes = imgPommes.Mirror(horizontally=True)

        # Conversion des images en bitmap pour les afficher
        bmpPommes   = wx.StaticBitmap(panel, wx.ID_ANY,
                                      wx.BitmapFromImage(imgPommes))
        bmpAbricots = wx.StaticBitmap(panel, wx.ID_ANY,
                                      wx.BitmapFromImage(imgAbricots))

        # creation de la zone de dessin
        monDessin = ZoneDessin(panel)

        sizer.Add(bmpPommes, pos=(0,0))
        sizer.Add(bmpAbricots, pos=(0,1))
        sizer.Add(monDessin, pos=(1,0), span=(1,2))

        sizer.AddGrowableCol(1)
        sizer.AddGrowableRow(1)

        panel.SetSizer(sizer)
        panel.Fit()
        self.Fit()


class ZoneDessin(wx.Panel):
    """ Zone de dessin """

    def __init__(self, parent):
        wx.Panel.__init__(self, parent, wx.ID_ANY, size=(510, 80))
        # recuperation de l'evenement EVT_PAINT
        wx.EVT_PAINT(self, self.OnDessiner)

    def OnDessiner(self, event=None):
        dc = wx.PaintDC(self)
        dc.Clear()
        # definition de la couleur et de la taille du pinceau
        dc.SetPen(wx.Pen("RED", 3))
        # dessine la ligne de la flèche
        dc.DrawLine(60, 50, 60, 7)
        # dessine la tete de la flèche
        dc.DrawPolygon([(55, 7),(65,7),(60,0)], fillStyle=wx.WINDING_RULE)
        dc.DrawText("les pommes, c'est bon", 70, 20)

        dc.SetPen(wx.Pen("GREEN", 3))
        dc.DrawLine(495, 50, 495, 7)
```

```
        dc.DrawPolygon([(490, 7),(500,7),(495,0)], fillStyle=wx.WINDING_RULE)
        dc.DrawText("les abricots, c'est meilleur!", 300, 30)

if __name__ == "__main__":
    app = wx.PySimpleApp()
    frame = Fenetre("Images")
    frame.Show()
    app.MainLoop()
```

11.12. Exemple d'utilisation des menus

```
#! /usr/bin/pythonw
# -*- coding: utf-8 -*-
import wx

class MainApp(wx.App):
    def OnInit(self):
        # Création de la fenêtre principale
        frame = MainWindow('Exemple de menu')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Fenêtre pour l'exemple de création de menus """
    def __init__(self, title):
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(300, 150))
        panel = wx.Panel(self)

        # simple label pour afficher les actions des menus
        self.texte = wx.StaticText(panel, wx.ID_ANY, "Aucune Action", pos=(30, 50))

        # Creation de la bar des menus
        self.__createMenuBar()

        self.Show(True)

    def __createMenuBar (self):
        """ methode privée qui va s'occuper de créer la barre de menu """

        menubar = wx.MenuBar()
        menuFichier = wx.Menu()
        menuEdition = wx.Menu()

        # création et ajouts des elements du menu Fichier
        mnuItemOuvrir = menuFichier.Append(wx.ID_ANY, "&Ouvrir\tCtrl-O")
        menuFichier.AppendSeparator()
        mnuItemQuitter = menuFichier.Append(wx.ID_ANY, "&Quitter\tCtrl-Q")

        # Sous Mac OS X, l'entrée Quitter de la bar de menu ne fait pas partie du
        # menu Fichier
        if "__WXMAC__" in wx.PlatformInfo:
            wx.App.SetMacExitMenuItemId(mnuItemQuitter.GetId())

        # création des elements du menu Edition
        mnuItemAnnuler  = wx.MenuItem(menuEdition, wx.ID_ANY, "Annuler\tCtrl-Z")
        mnuItemRetablir = wx.MenuItem(menuEdition,wx.ID_ANY,
                                      "Retablir\tCtrl+Shift-Z")

        # creation d'un sous menu
        sousMenuAction  = wx.Menu();
        mnuItemAction1 = sousMenuAction.Append(wx.ID_ANY, "Action 1")
        mnuItemAction2 = sousMenuAction.Append(wx.ID_ANY, "Action 2")
        mnuItemAction3 = sousMenuAction.Append(wx.ID_ANY, "Action 3")

        # Ajout des entrées au menu Edition
        menuEdition.AppendItem(mnuItemAnnuler)
        menuEdition.AppendItem(mnuItemRetablir)
        menuEdition.AppendSeparator()
        menuEdition.AppendMenu(wx.ID_ANY, "Action", sousMenuAction) # ajout du sous
                                                                    # menu
```

```
        # Ajout des menus a la barre de menu
        menubar.Append(menuFichier, "Fichier")
        menubar.Append(menuEdition, "Edition")
        self.SetMenuBar(menubar)

        # Liaison des evenements
        self.Bind(wx.EVT_MENU, self.OnOuvrir,  mnuItemOuvrir)
        self.Bind(wx.EVT_MENU, self.OnQuitter, mnuItemQuitter)
        self.Bind(wx.EVT_MENU, self.OnAction, mnuItemAction1)
        self.Bind(wx.EVT_MENU, self.OnAction, mnuItemAction2)
        self.Bind(wx.EVT_MENU, self.OnAction, mnuItemAction3)
        self.Bind(wx.EVT_MENU, self.OnNonImplemente, mnuItemAnnuler)
        self.Bind(wx.EVT_MENU, self.OnNonImplemente, mnuItemRetablir)


    def OnQuitter (self, evt):
        wx.MessageBox("Bye, bye!")
        self.Close()

    def OnOuvrir (self, evt):
        self.texte.SetLabel( "Appel de l'action Ouvrir du menu" )

    def OnAction (self, evt):
        itemAction = self.GetMenuBar().FindItemById( evt.GetId())
        label = itemAction.GetText()
        self.texte.SetLabel ("Appel de l'action %s du menu" % label)

    def OnNonImplemente (self, evt):
        self.texte.SetLabel ("Action non implementee")

if __name__ == "__main__":
    app = MainApp(redirect=False)
    app.MainLoop()
```

12. Annexes

12.1. Annexe A : Comparaison entre wxPython et PyQT

Dans la partie qui suit nous allons tenter d'effectuer un petit comparatif objectif entre les librairies wxPython et PyQt. Ces deux librairies permettent de créer facilement des interfaces graphiques en Python, mais en se basant sur des principes différents.

La première différence que l'on remarque entre les librairies Qt et wxWidgets (les libraires C++ de base) concerne la documentation de ces librairies. En effet, Qt semble plus professionnelle car il y a une société derrière, tandis que wxWidgets est une librairie communautaire. Par conséquent, la documentation de Qt est bien plus fournie et complète que celle de wxWidgets. Néanmoins, ni PyQt ni wxPython ne disposent à l'heure actuelle de véritable documentation de référence, car il est extrêmement simple de passer de C++ à Python. Il faut toutefois noter que la documentation de wxWidgets comporte certaines informations sur wxPython, lorsque par exemple les méthodes de wxPython sont quelque peu différentes de celles en C++. De plus, il existe un livre dédié entièrement à wxPython.

Qt dispose d'un système de gestion des événements extrêmement puissant et pratique. En effet, le système de signaux et slots permet d'affecter à un événement (signal) de multiples actions (slot), et une action peut être activée par de multiples événements. WxPython ne permet malheureusement pas, pour un même événement, d'appeler plusieurs méthodes.

À l'inverse de wxWidgets qui n'est qu'une librairie d'interface, Qt propose un ensemble de classes autre que la gestion d'interfaces. Citons entre autre les bases de données, la gestion de XML, la possibilité de faire des connexions réseaux, etc. Qt est donc bien plus qu'une simple librairie d'interface. Cependant, elle garde sa simplicité d'utilisation.

wxWidgets met à disposition du programmeur un système très performant concernant le placement des éléments graphiques. Les sizers, qui sont un peu difficile à appréhender au départ, permettent de gérer facilement le positionnement et le redimensionnement des éléments, en définissant par exemple que les éléments doivent garder leurs proportions, que certains éléments peuvent s'agrandir ou se rétrécir jusqu'à une certaine limite, etc. De telles possibilités existent aussi en Qt, cependant avec l'utilisation du Qt Designer, il devient très simple de créer et de tester l'interface d'une application.

Nous avons aussi constaté que l'installation de wxPython semble plus simple que PyQt. Cela est certainement dû au fait que la librairie Qt est beaucoup plus bas niveau que wxWidgets. Qt dispose de ces propres routines d'affichage, tandis que wxWidgets se base sur ce qui est fourni par le système d'exploitation sur lequel il tourne.

Néanmoins, pour conclure ce petit comparatif, il ne semble pas qu'une librairie soit meilleure que l'autre. Tout dépend de ce que l'on désire faire. Pour une application avec uniquement une interface graphique la librairie wxPython ira très bien, cependant si on désire faire une application plus complète, avec par exemple la gestion du réseau, PyQt sera un meilleur choix. Notez cependant qu'il est tout à fait possible de créer une application interagissant avec le réseau en wxPython, Python disposant de classes permettant de gérer le réseau. Le seul avantage de Qt sur ce point et qu'il n'y a besoin que d'une seule librairie. Avec les inconvénients que cela génère, comme la lourdeur du chargement, même si Qt en version 4 résoud ce problème en découpant la librairie en modules.

12.2. Annexe B : Script Python pour OpenOffice 2

En complément, nous donnons le code source du script "macro OO2.org" qui a permit de mettre en gras les mots-clés dans les codes Python de ce document.

```
# -*- coding: utf-8 -*-
from com.sun.star.awt.FontWeight import BOLD

# Utilisation d'un dictionnaire pour avoir une table
# de hachage dont les clés sont les mots réservé en
# Python. Les valeurs numériques sont arbitraires.
keywords_dic = { "and":1,     "assert":2,   "break":3,
                 "class":4,   "continue":5, "def":6,
                 "del":7,     "elif":8,     "else":9,
                 "except":10, "exec":11,    "for":12,
                 "from":13,   "finally":14, "global":15,
                 "if":16,     "import":17,  "in":18,
                 "is":19,     "lambda":20,  "not":21,
                 "or":22,     "pass":23,    "print":24,
                 "raise":25,  "return":26,  "try":27,
                 "while":28,  "yield":29 }

def parsePython():
   """ Effectue une analyse syntaxique Python sur des paragraphes
       donnés
   """
   # Permet de récupérer la représentation du document en cours
   # d'édition
   document = XSCRIPTCONTEXT.getDocument()

   # Permet d'obtenir la représentation textuelles et stylisée
   # du document.
   document_text = document.getText()

   # Création d'un curseur pour naviguer dans le document
   cursor = document_text.createTextCursor()

   # Parcours les paragraphes un à un
   while cursor.gotoNextParagraph(False):

      # Seul les paragraphes qui sont de style "Code_Source_Bordure" doivent
      # être analysés
      if cursor.getPropertyValue("ParaStyleName") == u'Code_Source_Bordure':

         # Analyse le paragraphe courant mot à mot
         while cursor.gotoEndOfWord(True):

            # Vérifie si le mot courant est un mot clé Python.
            if keywords_dic.has_key(cursor.getString().lstrip()):
               cursor.setPropertyValue("CharWeight", BOLD)

            cursor.gotoNextWord(False)

# Spécifie à OpenOffice les méthodes qu'il peut exécuter directement
g_exportedScripts = parsePython,
```

Attention: L'exécution de ce script sur des gros documents, tel que celui-ci, mène parfois à un plantage d'OpenOffice.org 2.