Bevezetés a wxPythonba

Jeremy Berthet & Gilles Doge
HEIG-VD, 2006. július 7.

Tartalomjegyzék

1. Bevezetés

A dokumentumot, amit az olvasó a kezében tart, eredetileg a HEIG-VD-ben (Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud) egy egy trimeszteres projekt keretében írták.

A projekt célja a wxPython könyvtár bemutatása annak működése, valamint az általa nyújtott funkciók leírása révén.

Az elképzelés szerint ezt a dokumentumot egy olyan – az elsajátított fogalmak bemutatására Python-t és wxPython-t alkalmazó – user interface kurzusnak kellene követni, mint amilyet a HEIG-VD-ben az "Informatika – Programozás" szakirány 2. évében tartanak.

1.1. Általános bemutatás

A wxPython API lehetővé teszi alap és fejlett felhasználói interface-ek készítését. Különböző objektumokat kínál a felhasználónak, amikkel egy modern alkalmazásban várhatóan találkozunk : kétdimenziós rajzolási lehetőséget, és az MDI (Multiple Document Interface) típusú interface-ek támogatását.

A wxPython egy egyszerű alkalmazói API, amit arra terveztek, hogy a Python nyelvvel használják. Egy másik API, a wxWidgets, adaptációjáról van szó, amit eredetileg a C++ nyelvhez terveztek.

Ennek az a következménye, hogy a két API szerkezete (a class-ok hierarchiája, a metódusok nevei, stb.) majdnem pontosan megegyező és az, hogy egyszerűen hivatkozni lehet a wxWidgets nagyszámban fellelhető dokumentációjára.

2. Telepítés

2.1. Linux/Unix

A Linux disztribúciók többségéhez léteznek package-ek, amik mindent elvégeznek a felhasználó helyett. RPM-ek letölthetők a wxPython weboldaláról és a libwxgtkX.Y-python csomagok benne vannak a Debian depóban (X és Y a verziószámok).

Egyébként a wxPython függ a **glib** és a **gtk+** könyvtáraktól, amik alapértelmezetten telepítve vannak egy GNOME környezetben.

2.2. Windows

A wxPython site -on a Python 2.3, 2.4, 2.5-höz megtalálhatók a wxPython könyvtár telepítő programjainak Unicode illetve ANSI verziói.

2.3. MacOS 9 és MacOS X

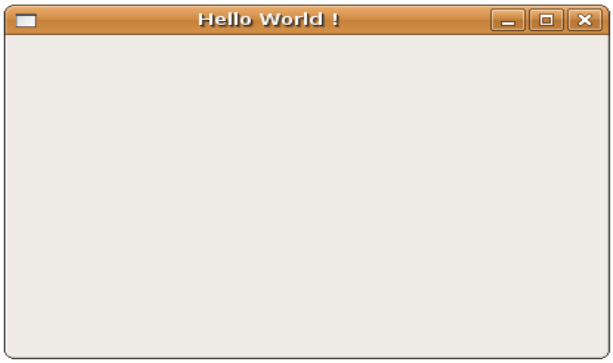
A MacOS X 10.3 (Panther) óta (a Python 2.3 -tól kezdve) a wxPython könyvtár alapértelmezetten telepítve van, így semmilyen különönleges telepítést nem igényel. A wxPython script-et a pythonw interpreterrel kell indítani eltérően egy normál Python-programtól. Amennyiben egy korábbi MacOS verziója van, úgy a wxPython-t külső csomagokból kell telepíteni.

3. Alapfogások

3.1. Hello World !

A tanulás általában a lehető legegyszerűbb példával kezdődik. Mi sem fogunk kivételt tenni e szabály alól. Meg lehet nyitni az általunk preferált szövegszerkesztővel a "hello.py" nevű első példaprogramot és végre lehet hajtatni.

A következő ábrához hasonló képet kell kapnunk (az alkalmazott operációs rendszertől függően lehetnek kisebb eltérések).



ábra 1: Hello World !

Ha a program nem hajtódik végre, akkor nézzük meg a telepítés rendszerünkre vonatkozó fejezetét és nézzük meg a dokumentációt.

Megállapíthatjuk, hogy nem volt szükség sok kódsorra. Ráadásul az első példánkat túl struktúráltuk, amikor két class-ba osztottuk szét a kódot, ami nem feltétlenül szükséges.

Elemezzünk néhány fontos sort :

```
import wx
```

Minden wxPython alkalmazásban legalább ez az import utasítás szerepelni fog. Arról van szó, hogy a modulbeli class-okat importáljuk az alkalmazásunkba azzal a céllal, hogy használhassuk őket.

A MainWindow class a **wx.Frame** alapclass leszármazottja, mely utóbbi egy ablakot reprezentál a képernyőn. A class konstruktora a szülőclass-a konstruktorát hívja paraméterekkel, amik közül egyeseknek az értelmét már bizonyára kitalálta az olvasó.

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

A Show metódust a wx.Frame -től örökölte és ez teszi lehetővé, hogy megmondjuk az ablaknak, hogy a megadott paraméternek megfelelően kiíródjon-e vagy sem.

A wxPython a speciális **wx.App** class-t (valójában több is rendelkezésre áll) használja az alkalmazás együttesének a definiálására. Ez a 'becsomagolás' lehetővé teszi a grafikus interface-ek bizonyos speciális konstrukcióinak egyszerűbb kezelését.

Példánkban a **MainApp** class ebből származik és létrehozza a főablakunkat reprezentáló objektumot. wxPython-specifikus szervezési okokból a konstruktor helyett az **OnInit** metódust kell átdefiniálni. Ez a metódus létrehozza az ablakunk egy példányát és kiíratja az előtérben.

Végül : az alkalmazás-objektumunk csak abban az esetben jön létre, ha ez a script a főprogram. A **MainLoop** hívását az eseménykezelésre vonatkozó részben fogom elmagyarázni.

3.2. A wx.App class

Az előző alfejezetben egy olyan programot láttunk, amit két class-ból hoztunk létre. Ez a két class nem véletlenül van. Az **App** és a **Frame** class-ok minden wxPython alkalmazásnak alapelemei, melyekből a program többi része származik.

Bár közvetlenül ebből a két osztályból létrehozhatók objektumok, a gyakorlat azt igényli, hogy mindegyik objektumot úgy származtassuk le, ahogyan azt az első programpéldánkban tettük. A Frame class-t a 6. fejezetben fogjuk részletesen leírni. Most az **App** class-t fejtjük ki részletesebben.

Egy wxPython alkalmazásban lennie kell egy (és kizárólag csak egy) **App** objektumnak, még mielőtt a legcsekélyebb grafikus komponens példányt (a priori a főablakot) létrehozzuk. Lényegében ez a különleges objektum hat közvetlenül kölcsön a rendszerrel és ez az az objektum, ami előkészíti a terepet a grafikus objektumaink létrehozása számára.

Azért nem tanácsos új konstruktort definiálni az **App** class-ból leszármaztatott class-ok számára, mert azt kockáztatjuk, hogy azzal megzavarjuk a class-ok belső működését. Ez az oka annak, hogy az App class-nak van egy **OnInit()** metódusa, aminek a hívására az objektum létrehozásakor automatikusan sor kerül.

Azért, hogy biztosak legyünk benne, hogy nem hozunk létre grafikus komponenst azelőtt, hogy egy App objektumot generálunk, azt a szokást fogjuk fölvenni, hogy az ablak-objektumokat az **OnInit()** metódusban hozzuk létre. Innen van az értelme annak, hogy mindig létrehozunk egy App subclass-t, még akkor is, ha a subclass nem terjeszti ki szülő-class funkcióit.

Azon túl, hogy a rendszert felkészíti grafikus komponens létrehozására, az App objektumnak van egy eseménykezelője, ami a **MainLoop()** metódus hívásakor indul el. Alapjában véve az eseménykezelő egy végtelen ciklus, ami két dolgot művel.

Először is az operációs rendszer felől jövő jelekre vár (például egy egér-klikkelésre), majd amikor egy ilyen jelet fogadott, ezt különböző paraméterekkel (az esemény típusa, az esemény forrása, stb.) hozzáadja egy várakozási sorhoz.

Utána periódikusan kivesz egy eseményt a várakozási sorból, majd hívja az általa olvasható paramétereknek megfelelően kapcsolt metódust. Ha semmilyen metódus sincs kapcsolva, akkor elfelejti az eseményt és a következőt veszi ki a várakozási sorból.

Ezért amit a programozónak meg kell tenni az az, hogy hozzá kell kapcsolni egy metódust egy eseményhez, amint azt a következő alfejezetben látni fogjuk.

Az App class még számos dolgot lehetővé tesz, amik leírása meghaladja ennek a jegyzetnek a kereteit. További információkat a "wxPython in action" című könyvből lehet szerezni.

Végül jegyezzük meg, hogy az eseménykezelő - jól lehet azt mondjuk, hogy egy végtelen

ciklus - megszakad, ha a programban nincs több Frame-objektum. Következésként az App objektum megsemmisül és nem tudunk többé grafikus komponenst generálni azelőtt, hogy egy App objektumot újra létrehoznánk.

3.3. Widget / kód kapcsolat

Amikor "valami történik egy Python-objektummal", akkor egy esemény generálódik. Ennek a "valaminek" a természete igen eltérő lehet. Létrehozhatja a felhasználó például egy egérgombbal való kattintással, vagy közvetlenül generálhatja egy objektum, amikor megváltoztatta az állapotát.

Láttuk, hogy az eseményeket, ha már egyszer létrejöttek, egy eseménykezelő fogja el és az amit tennünk kellett, az az volt, hogy hozzájuk kellett kapcsolnunk azokat az akciókat, amiket akkor kell végrehajtani, amikor az eseménykezelő egy meghatározott eseményt detektál. A goodbye.py program egy rövid példa a megvalósításra.

Most nem időzünk el egy Panel és egy nyomógom generálásánál – ezt később fogjuk leírni -, hanem nézzük meg a **`Bind()`** kulcsmetódust.

Ez a metódus az **`EvtHandler`** class-ban van definiálva, amiből számos class van leszármaztatva (nevezetesen a **`Frame`** és a **`Menu`** class-ok, valamint az **`App`** class, amit már bizonyára kitalált az olvasó). Ez lehetővé teszi, a kapcsolatteremtést egy eseménytípus, egy látható metódus és egy látható widget között. Látható a kód láthatóságának értelmében, nem pedig abban az értelemben, hogy az objektum ki van-e íratva vagy sem a képernyőre.

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

A **`wx.EVT_BUTTON`** egyike a rendelkezésünkre álló, a bekövetkezhető eseményeket reprezentáló, nagyszámú konstansnak. Az előbbi konstans egy egérkattintást reprezentál a frame egyik gombján (ez az oka annak, hogy a frame **`Bind()`** metódusát hívjuk). A wxPython on-line dokumentációjában az Event class és subclass-ainak tanulmányozása során a különböző widget-ekhez megtaláljuk az események listáját.

Ebben az illusztratív példában van egy **`OnClick()`** metódusunk a **`MainWindow`** class-ban, ami az alkalmazásban a **`Close()`** függvény (amit később fogunk elmagyarázni) segítségével történő bezárást igényli.

Ennek a metódusnak bármilyen neve lehet, például **`ClickOnButton()`**.

A metódusnak kell, hogy legyen egy paramétere, ami az eseményobjektumot fogja tartalmazni. Egy class-metódus esetében ennek a paraméternek a másodiknak kell lenni (tekintettel arra, hogy az első paraméter a **`self`**) vagy az elsőnek kell lenni, ha egy "statikus" függvényről van szó. Ezt a paramétert hagyományosan **`event`** -nek fogjuk nevezni.

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

Jegyezzük meg, hogy egy másik metódus paramétereként az **`OnClick()`** metódus egy példányát adjuk át, tehát nem kell kitenni a zárójeleket !

Végül az utolsó paraméter a kapcsolandó esemény forrásobjektumának azon példánya, melyet

akkor kell kapcsolni, ha a **Bind()** hívása nem a közvetlenül az eseményt kiváltó objektumon történik, mint ebben az esetben. (Az **EVT_BUTTON** esemény a frame-től ered, de nem adja meg, hogy melyik gombon történt a kattintás.)

Íme, ezzel az egyetlen sorral összekapcsoltunk egy metódust, ami befejezi a programot, egy megadott egérgombbal történő kattintással. Ez az eljárás minden olyan objektumra általánosítható, aminek lehetnek kapcsolható eseményei.

A **Bind()** a forrásobjektum helyett elfogadhat két paramétert - id1 és id2 – is. Ez lehetővé teszi, hogy ugyanazt az eseményt hozzárendeljük egy olyan objektumcsoporthoz, mely objektumait egymásután következő id-k azonosítják. A **Bind()** metódust ennek ellenére ritkán hívjuk ilyen módon. Általában preferálandó az id-k használatának kerülése. (A wxPython-ban minden objektumnak van egy id nevű tulajdonsága, ami egy egyedi számérték.)

Végül : a forrásparaméter (és az id1, id2 paraméterek) opcionálisak. A saját felelősségére alkalmazza a programozó !

3.4. Widget-ek közös tulajdonságai

A wxPython-ban minden megjeleníthető objektum egy szülőosztályból, a wx.Window -ból van leszármaztatva.

Ennek az osztálynak a konstruktora a következő alakú :

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

,ami lehetővé teszi minden megjeleníthető objektum számára közös tulajdonságok definiálását.

A **parent** paraméter. amint azt a neve is jelzi, az általunk létrehozott objektum szülőobjektumát definiálja. Ez lehetővé teszi egy objektum-hierarchia megőrzését, mely hierarchiában az egyik objektum tartalmazza a másikat.

Ha megint elővesszük az előző kódot, láthatjuk, hogy az ablak (wx.Frame) tartalmaz egy konténerobjektumot (a wx.Panelt -t, aminek a wx.Frame a szülője), ez utóbbi pedig egy gombot tartalmaz.

Nyilván való, hogy a következő metódusokkal lehetőség van az így létrehozott fában való mozgásra :

```
GetParent()
```

Ez a metódus teszi lehetővé a szülőobjektumra való hivatkozást.

```
GetChildren()
```

Ezzel a metódussal az objektum gyermekobjektumainak hivatkozásait tartalmazó listát kapjuk meg. Lényeges, hogy tudjuk : ez a lista másolata a wxPython által létrehozott listának, tehát egy új gyermekobjektum létrehozása nem frissíti.

Ez érdeklődésre tarthat számot egyes alkalmazások esetében vagy objektumoknak futásközben egy panelen történő pozícionálásakor. Az olvasón a sor, hogy kreatív legyen.

A fastruktúrán kívül a wxPython minden widget számára fenntart egy EGYEDI azonosító

számot. Ha két objektumnak megegyezik az azonosítója, az az alkalmazás nem várt viselkedését okozhatja.

Szóval milyen értéket adjunk az id paraméternek ?

Ha nem számítunk ennek a számnak az explicit használatára, akkor inkább hagyjuk meg az alapértelmezett értéket és a konstruktor paramétereit névvel adjuk át, vagy pedig a **wx.ID_ANY** értéket használjuk, ha a paramétereket pozíciójuk alapján adjuk át. A wxPython tehát automatikusan magára fogja vállalni egy ID generálását a widget-ünk számára.

Ha nem, akkor lehetőség van a **wx.NewId()** függvény segítségével egy id generálására egy egyedi ID-nek egy változóba történő bejegyzéséhez és ennek a változónak a konstruktor paramétereként történő utólagos átadására.

Végezetül, bármelyik legyen az alkalmazott módszer, mindegyik widget-nek van egy GetId() metódusa ha minden kötél szakad.

A konstruktor **pos** és **size** paraméterei megadják az objektum pozícióját és kezdeti méretét. A **DefaultPosition** és a **DefaultSize** konstansok az éppen létrehozott objektumtól függnek.

Bár a wxPython definiálja a **wx.Point** és **wx.Size** class-okat a megfelelő metódusokkal, általában egyszerűbb egy két egész számból képzett tuple-t használni egy objektum helyének és méretének megadásához. Azt a goodby.py példában megállapíthatjuk.

A wx.Window, de számos belőle leszármaztatott widget is definiál olyan stílusokat, amik befolyásolhatják az objektum megjelenését a képernyőn. Ezek azok a konstansok, amik minden widgetre ki vannak listázva a Reference manual-ban.

Ezek a konstansok egy-egy atomi stílust definiálnak. Ha több stílust akarunk keverni, akkor használhatjuk a | (logikai VAGY) operátort. Az inverz művelet is lehetséges, a ^ operátor segítségével. ami stíluselemeket távolít el egy atomi stílusokból álló stíluskeverékből.

Még mindig a goodby.py példánknál maradva, az ablakgeneráló operátorok egy alkalmazási példáját láthatjuk.

Ugyanakkor ügyeljünk két dologra. A Reference manual még távolról sem teljes, ezért egy internetes keresés néha informatívabb a rendelkezésre álló stílusokra vonatkozóan. A wxPython in action könyv is jó ötleteket ad.

Az operációs rendszertől függően a paraméterek hatása nagyon eltérő lehet. Ez kellemetlen lehet a **pos** paraméter alkalmazásakor. A későbbiekben látni fogunk egy fejlett pozícionálási módszert.

Végezetül, a **name** paraméter lehetővé teszi a névadást az objektumnak, azonban ez a paraméter látszik különösebben hasznosnak.

Már ismerjük a wxPython működésének alapjait. A későbbiekben látni fogjuk a grafikus alkalmazásainkban használható különböző objektumokat.

4. Ablakozás

4.1. A Frame és a Top-Level Window fogalma

A wxPythonban egy grafikus interface legalább egy "top-level window objektumnak" nevezett objektumot tartalmaz, ami általában egy wx.Frame (ablak) típusú objektum. Ezt az objektumot nem tartalmazza egy másik widget, ez az objektum a grafikus alkalmazás egyfajta "gyökere". Általában ez az az objektum, amit a grafikus alkalmazásunk főablakaként definiálunk és ami a többi grafikus objektumot tartalmazza.

Több olyan objektumunk lehet, amiknek nincs szülőobjektumuk (több top-level window-nk lehet), minden esetre meg kell adjuk a wxPythonnak, hogy közülük melyiket fogjuk a fő top-level window-nak tekinteni. Erre való a **`SetTopWindow()`** metódus. Ha nem adjuk meg explicit módon, hogy melyik frame lesz a fő top-level window, akkor a wxPython a **`wx.App`** -ban elsőként definiált frame-et fogja annak venni.

4.2. Kilépés az alkalmazásból

Az alkalmazásból akkor lépünk ki, ha minden top-level window-t bezártunk, vagyis ha minden szülőobjektum nélküli ablak zárva van, nem csak a **`SetTopWindow()`** metódussal kijelölt ablak.

Egy frame-et bezárhatunk a **`Close()`** metódus hívásával. Az alkalmazásból való kilépés egy másik módja a **`wx.Exit()`** globális függvény használata. A **`wx.App`** class **`OnExit()`** metódusának köszönhetően definiálhatunk egy "takarítást" pontosan az azt megelőző pillanatra, hogy kilépünk az alkalmazásból. Ennek a metódusnak a hívására akkor kerül sor, amikor "top-level window-t" bezártuk. Ezt felhasználhatjuk például hálózati kapcsolatok vagy egy adatbáziskapcsolat zárására.

4.3. Pozícionálás

A grafikus elemeknek egy ablakban történő pozícionálására két eltérő megközelítés létezik. Az első az abszolút pozícionálást alkalmazza, míg a második egy speciális – Sizer-nek nevezett – elemet használ.

4.3.1. Abszolút pozícionálás

Az abszolút pozícionálás különböző objektumok elhelyezésére szolgáló nagyon egyszerű és intuitív eljárás. Elegendő az (X, Y) koordinátákat használni, az origó a frame balfelső sarkában van. Ez az eljárás nagyon jól működik akkor, amikor teles ellenőrzésünk alatt áll az ablak mérete és az ablak által tartalmazott widget-ek száma. Minden esetre hamar elérjük ennek a megközelítésnek a korlátait, nevezetesen akkor, amikor az ablak mérete nem rögzített. Ebben az esetben a Sizer-ek használatát részesítjük előnyben.

4.4. Sizer-ek

4.4.1. Pozícionálás sizer-ek segítségével

A Sizer egy objektum, aminek az az egyetlen feladata, hogy kezelje egy widgethalmaz elrendezését egy konténerben. A Sizer se nem konténer, se nem widget. Valójában egy pozícionáló algoritmus reprezentációja. Minden Sizer a **`wx.Sizer`** absztrakt class leszármazottja.

A wxPython-ban ötféle Sizer van :

Sizer típusa	Leírás
Grid	Egy egyszerű, szabályos rács (mindegyik cellájának azonos a mérete). Kevéssé tipikus alkalmazása : egyszerű játék táblájaként.
Flex Grid	Rugalmas rács, ami lehetővé teszi a jobb elrendezést, amikor a widget-ek különböző méretűek.
Grid Bag	A legflexibilisebb sizer. A widget-ek tetszőéeges elrendezését engedi meg a rács belsejében
Box	Hasznos a widget-ek vízszintes és függőleges elhelyezése során.
Static Box	Lehetővé teszi egy widget-csoport bekeretezését. Szöveg hozzáadására is van lehetőség.

Egy sizer alkalmazása három lépésből áll :

1. Létrehozzuk a sizer-t egy konténerben. Egy sizer-t a **`wx.Window`** (minden megjeleníthető widget super-class-a) objektum **`SetSizer(sizer)`** metódusával teszünk bele egy konténerwidget-be.
2. A **`wx.Sizer Add()`** metódusával mindegyik gyermek-widget-et hozzákapcsoljuk a sizer-hez.
3. (Opcionálisan) aktiváljuk a sizer automatikus méretbeállítását (a **`wx.Window Fit(window)`** metódusának többszörös hívásával).

4.4.2. Widget-ek hozzákapcsolása a sizer-hez

A widget-eknek a sizer-hez való kapcsolására szolgáló legmegszokottabb eljárás az **Add()** metódus alkalmazása. Ez a sizer gyermek-widget-jei listájának **végéhez** hozzáadja a widget-et. Az **Add()** metódusnak három változata van :

```
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)
```

Az első változat a leggyakrabban használatos, lehetővé teszi egy widget (a window argumentum) hozzáadását a sizerhez.
A második változat lehetővé teszi a sizerek egymásba ágyazását.
A harmadik változatot a size (**wx.Size** típusú) argumentummal definiált méretű üres terület hozzáadására használjuk.

A többi paraméter a widget elrendezésének módját fogja befolyásolni. Számos paraméter csak bizonyos sizer-ek esetében áll rendelkezésre. A **proportion** argumentumot csak a boksz sizer használja. Ez a paraméter adja meg, hogy hogyan lesz újraméretezhető az elem, amikor a konténere megváltoztatja a méretét.
A **flag** argumentummal lehet a kiíratási opciókat megadni (mint az elrendezést, a keretet vagy az újraméretezést). A **border** argumentum határozza meg a keret méretét, ha a **flag** paraméterrel keretet definiáltunk. Végezetül, a **userData** paramétert abban az esetben használhatjuk, amikor saját sizer-t definiálunk.
A flag paraméterben átadandó különböző konstansok megtalálhatók az API dokumentációjában a wx.Sizer Add metódusának leírása alatt.
Léteznek más metódusok is a widget-ek sizer-hez kapcsolására. Az **Insert()** metódus egy pozícióindex megadásával kapcsol a sizer-hez egy widget-et, a **Prepend()** metódus azonos az **Add**() -dal, az a különbség a két metódus között, hogy az előbbi a lista elejéhez adja a widget-et. További információk a wxPython dokumentációjában találhatók.

4.4.3. Widget-ek törlése a sizer-ből

A **Detach()** metódussal lehet leválasztani egy widget-et a sizer-ről. Ez a metódus nem rombolja le az objektumot !

Három változata van :

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

Ez a metódus mindhárom esetben egy boolean értéket ad visszatérési értékül. A visszatérési érték false, ha olyan elemet próbálunk meg törölni, ami nincs a sizer-hez kapcsolva. Az első utasítás egy widget-et, a második egy sizer-t, a harmadik a paraméterben megadott indexértéknek megfelelő elemet törli.

Egy elem leválasztása a sizer-ről nem fogja megváltoztatni az automatikus kiíratást. Ehhez a **Layout()** metódust kell hívni.

4.5. A Grid Sizer

A legegyszerűbb sizer, amint a neve is jelzi. Egy kétdimenziós rácsról van szó melyben a gyermek-widget-ek balról jobbra és felülről lefelé vannak elhelyezve. Tudni kell, hogy a sizer-nek van egy belső listája a gyermek-widget-ekről. Ennek a listának az első widget-e a balfelső sarokban fog elhelyezkedni.

Amikor átméretezünk egy grid sizer-t, akkor minden cella mérete azonos módon fog megváltozni. Alapértelmezetten ez a méret a rácsbeli legnagyobb widget méretén alapul és a gyermek-widget-ek mérete nem alkalmazkodik az előbbihez.
A Grid Sizer konstruktorának a következő a prototípusa :

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

Ahol :

- **rows** : a rács sorainak száma
- **cols** : a rács oszlopainak száma
- **vgap** : az oszlopok közötti függőleges sáv
- **hgap** : az oszlopok közötti vízszintes sáv

Ha a **`rows`** vagy a **`cols`** értéke 0, akkor a sorszám vagy az oszlopszám a sizer-hez kapcsolt widget-ek számától fog függeni. Ha például a sizer-t a következő paraméterértékekkel hoztuk létre : `wx.GridSizer(2, 0, 0, 0)` és a sizer-hez 8 widget van kapcsolva, akkor azok 2 sorban és 4 oszlopban lesznek.

A `rows`, `cols`, `vgap` és `hgap` tulajdonságok mindegyikének van setter és getter metódusa : **`GetRows(), SetRows(rows), GetCols(), SetCols(cols), GetVGap(), SetVGap(vgap), GetHGap(), SetHGap(hgap)`**.

4.6. A Flex Grid Sizer

Ez a sizer flexibilisebb, mint az előbb látott Grid Sizer. Az alábbiakban felsoroljuk a főbb különbségeket :

- Minden sor és oszlop méretét külön határozza meg.
- Alapértelmezetten nem változtatja meg automatikusan a cellái méretét amikor átméretezzük. Megadhatjuk, hogy egy méretváltozás során mely sorokat és mely oszlopokat fogja automatikusan újraméretezni.

Egy Flex Grid Sizer konstruktorának ugyanazok a paraméterei, mint a Grid Sizer-nek :

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

Annak definiálásához, hogy mely sorok vagy oszlopok "nőhetnek meg" a sizer méretének függvényében, a következő paramétereket kell a konstruktornak átadni :

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

Az első paraméter azt az oszlopot vagy sort adja mag, amelyiknek változó méretűvé kell válni (egy 0 és N-1 közé eső egész).

Amikor a sizer mérete vízszintesen nő, akkor alapértelmezetten úgy működik, hogy az új méret megoszlik a(z **`AddGrowableCol()`** metódussal) változó méretűnek definiált oszlopok között. Ugyanez vonatkozik a sorokra a sizer függőleges irányú méretváltozása esetén. A `proportion` paraméter használatával megváltoztathatjuk ezt az alapértelmezett viselkedést.

Például, ha a 2 változó méretű oszlopot definiáltunk és az arányaik 2 és1, akkor az első oszlop a rendelkezésre álló terület 2/3-át, míg a második oszlop csak az 1/3-át fogja elfoglalni.

Ha a változó méretű oszlopaink egyikére megadjuk az arányt, akkor minden változó méretű oszlop számára meg kell adni egy arányt. Ugyanez vonatkozik a sorokra.

4.7. A Grid Bag Sizer

A flex grid sizer továbbfejlesztett változata a grid bag sizer. Két lényeges újdonságot vezet be :

- Lehetőséget ad arra, hogy a rács egy meghatározott cellájába tegyünk egy widget-et.
- Lehetőség van a sorok vagy az oszlopok egyesítésére.

Egy Grid Bag Sizer konstruktorának nincs szüksége sorszámokra és oszlopszámokra, mert az objektumainkat egy sor- és egy oszlopszám direkt megadásával szúrjuk be.

A következő a prototípusa :

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

4.7.1. Egy widget hozzákapcsolása a wx.GridBagSizer-hez

A grid bag sizer Add() metódusa az előbb bemutatott sajátosságok miatt különbözik más sizer-ek Add() metódusaitól.

Három változata van, mint a **`wx.Sizer`** generikus **`Add()`** metódusának :

```
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)
```

Ami változott, az a `pos` és `span` paraméterek megjelenése. A `pos` paraméterrel lehet megadni, hogy hova fogjuk beszúrni a widget-et. Ez egy (sor, oszlop) tuple, a (0,0) hivatkozás a sizer balfelső sarkát jelenti. Például : a (0,2) tuple-t kell megadni ahhoz, hogy egy objektumot az első sorba és a harmadik oszlopba szúrjunk be.

A `span` paraméter azoknak az oszlopoknak és soroknak a számát reprezentálja, amiket egyesíteni kell. Ha például azt akarjuk, hogy a widget-ünk 3 sort és 2 oszlopot foglaljon el, akkor a (3,2) tuple-t kell megadnunk.

A függelékben található egy példa (gridbagsizer.py) a grid bag sizer alkalmazására.

4.8. Box Sizer

A box sizer egyetlen oszlop vagy sor, amiben a widget-ek egymás után felülről lefelé illetve balról jobbra vannak elhelyezve.

A konstruktornak csak egy paramétere van, ami az irányt határozza meg :

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

Az `orient` paraméter a `wx.VERTICAL` (függőleges elrendezés) vagy `wx.HORIZONTAL` értékeket veheti fel. Ezt követően nem változtathatjuk meg a sizer irányát, csak arra van lehetőség, hogy a **`GetOrientation()`** metódussal megtudjuk merre fele irányul.

4.9. A Static Box Sizer

A static box sizer egy box sizer-t és a static box objektumot kombinál. A static box objektum teszi lehetővé a widget-ek vízuális csoportosítását úgy, hogy szöveges kerettel veszi körül a widget-eket. A **`wx.StaticBox`** class teszi lehetővé ennek az objektumnak a létrehozását. Az ehhez a csoporthoz tartozó widget-eket nem lehet közvetlenül belevenni egy **`wx.StaticBox`** típusú objektumba, tehát létre kell hozni egy sizer-t. A kérdéses sizer-t **`wx.StaticBoxSizer`** -nek nevezik.

Egy példa :

```
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)
```

Először egy **`wx.StaticBox`** objektumot hozunk létre, amit össze fogunk kapcsolni egy **`wx.StaticBoxSizer`**-rel. Ezután hozzákapcsoljuk a widgeteket a sizer-hez.

4.10. Párbeszéddobozok (dialogbox-ok)

A wxPython az előre definiált párbeszéddobozok széles skáláját kínálja a felhasználónak. Ezek mind ugyanattól a **`wx.Dialog`** szülő-class-tól származnak.

4.10.1. Párbeszéddoboz egy egyszerű üzenettel

A felhasználóval történő kommunikációra szolgáló legegyszerűbb párbeszéddoboz a **`wx.MessageDialog`**. Ez egy egyszerű doboz egy üzenettel, mely esetében a kiírandó gombok konfigurálhatók. A következő kód egy modális ablakot hoz létre, ahol igennel vagy nemmel kell válaszolni.

```
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()
```

Az üzenetet tartalmazó párbeszéddoboz konstruktorának prototípusa a következő :

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

Itt a `parent` paraméter a szülőablakot jelenti (Nem mint ha a párbeszéddoboznak egy Top-Level window-nak kellene lenni.) A `message` paraméter a kiírandó karakterláncot tartalmazza és a `caption` paraméter az a cím, aminek az ablak címében meg kell jelenni. Jegyezzük meg, hogy MacOS X alatt, a `caption` karakterlánc nem jelenik meg az ablak címében, hanem az üzenet fölött félkövér betűstílussal íródik ki. A `style` paraméterrel definiálhatjuk, hogy mely gombok jelenjenek meg a modális ablakban, sőt előredefiniált ikonokat is kiírathatunk. Az utolsó – `pos` – paraméterrel adhatjuk meg, hogy a párbeszéddoboz hol jelenjen meg a képernyőn.

A **`ShowModal()`** metódus kiíratja a modális ablakot és visszatérési értékként megadja annak a gombnak az (int típusú) azonosítóját, amelyik gombra kattintottunk. Amikor a párbeszéddoboz kiíródik, akkor az alkalmazás többi ablaka inaktívvá válik. Ez a metódus minden párbeszéddoboz esetében azonos. A következő koránt sem kimerítő lista a visszatérési értékeit közül felsorol néhányat : `wx.ID_YES, wx.ID_NO, wx.ID_CANCEL, wx.ID_OK`.

Egy modális ablak stílusai a következők lehetnek : `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.`
Ezek a (| operátorral) a bitmaszk-elvnek megfelelően kombinálhatók.

4.10.2. Párbeszéddoboz adatbeírással

Egy másik párbeszéddoboz típus egy ablak, amiben van egy adatbeviteli mező. A wxPython ezt a **`wx.TextEntryDialog`** class-ból hozza létre :

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

A `defaultValue` az adatbeviteli mező alapértelmezett értéke. A **`GetValue()`** és **`SetValue(value)`** metódusokkal hozzáférhetünk az adatbeviteli mező tartalmához vagy módosíthatjuk azt.

4.10.3. Párbeszéddoboz egyszeres kiválasztással

Ez a párbeszéddoboz lehetővé teszi egy érték kiválasztását egy értékkészletből. Ezt az értékkészletet lista (listbox) reprezentálja. A listát kezelő class a **`wx.Sin`**`gle`**`ChoiceDialog`** :

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

A paraméterei megegyeznek a **`wx.TextEntryDialog`** konstruktorának paramétereivel, azzal a különbséggel, hogy egy karakterlánc helyett ez esetben egy karakterláncokból álló listát adunk meg.

Példa :

```
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()
```

A **`GetStringSelection()`** metódussal hozzáférhetünk a kiválasztott karakterlánchoz, míg a **`GetSelection()`** metódus a kiválasztott elem indexét adja meg. A **`SetSelection(index)`** metódussal is definiálhatunk egy kiválasztást.

4.10.4. Párbeszéddoboz többszörös kiválasztással

Az előző párbeszéddoboz csak egyetlen egy elem kiválasztását tette lehetővé a listából, a **`wx.MutliChoiceDialog`** több elem kiválasztását teszi lehetővé. A **`GetSelections()`** metódust használjuk a kiválasztott elemek indeexlistájának kinyerésére és a **`SetSelections(selections)`** metódust a kiválasztások definiálására !

4.10.5. Egyéb párbeszéddobozok

A wxPython könyvtár számos egyéb párbeszéddobozt kínál, mint amilyenek a file/folder megnyitására, színválasztásra, fontkészletválasztásra, keresésre-cserére, stb. szolgáló párbeszéd-dobozok.

A példákat tartalmazó folderben az olvasó találni fog dialog.py nevű scriptet, ami az előbbiekben látott modális ablakokat alkalmazza, valamint egy "színválasztót".

5. A fókusz kezelése

Ebben a fejezetben és a következőben a gestion_liste.py programpéldával foglalkozunk.

Az említett fejezetek bemutatják a felhasználó - program belsejében történő - követését támogató fókuszkezelési lehetőségeket. Több olyan alapwidget-et használnak, amiket a 8. fejezetben fogunk bemutatni.

Ebben a példaprogramban a felhasználó fókusszal történő követésének kezelése távolról sem tökéletes. Ha az olvasó gyakorolni akar, arra biztatjuk, tökéletesítse.

5.1. Fókuszesemények

Minen egyes alkalommal, amikor egy widget ilyen, vagy olyan módon megkapja a fókuszt, akkor generál egy `wx.EVT_SET_FOCUS` eseményt, ami igényünk szerint kezelhető.

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

A példában amikor a lista megkapja a fókuszt (tehát egy elemet kiválasztottunk), akkor a "Modifier" és "Supprimer" gombok aktiválva vannak. Így láthatunk a **`Bind()`** metódus alkalmazására egy olyan példát, ahol a metódust nem egy Frame, hanem egy más widget-en alkalmazzuk.

Szimmetrikusan, egy widget, ami elveszti a fókuszt, generál egy `wx.EVT_KILL_FOCUS` eseményt.

5.2. Manuális fókusz

Néha érdekes lehet a fókusz manuális átadása egy adott widget-nek, speciálisan bizonyos programbeli műveletek után.

A **`wx.Window`** class-nak (és következésként a belőle leszármaztatott kontroloknak) van egy **`SetFocus()`** metódusa, ami átadja a fókuszt a widget-nek és a föntebb említett `wx.EVT_SET_FOCUS` és `wx.EVT_KILL_FOCUS` eseményeket létrehozza.

5.3. Fókusz letiltása

Az **`Enable()`** metódus segítségével, aminek egyetlen boolean paramétere van, megakadályozhatjuk, hogy egy widget megkapja a fókuszt.

A widget inaktívvá válik, ha a boolean paraméter értéke `False` és többet nem kapja meg a fókuszt. Nyilvánvalóan ez az állapot törölhető a metódus `True` paraméterértékkel történő hívásával.

5.4. Navigáció tabulátorral

Alapértelmezetten csak egy **`wx.Panel`** vagy egy **`wx.Dialog`** típusú widget belsejében aktív a tabulátoros navigáció. Vannak rá eszközök, hogy aktiváljuk más konténerekben, de ezt a lehetőséget nem fogjuk tárgyalni ennek a dokumentumnak a keretében.

Ez az oka, hogy a példaprogramnak van egy **`wx.Panel`** objektuma, amibe bele van téve a fő-sizer.

Elvben a navigáció sorrendje megegyezik a **wx.Panel** gyermekobjektumainak Python kódon belüli létrehozásának sorrendjével.

A dinamikus widget-kreálás keretében vagy amikor a kódot világossá kell tenni, a következő metódusok segítségével egy widget-et egy másik widget elé vagy után lehet tenni a navigációs sorrendben.

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

A programpéldában az "Enregistrer" gombnak a lista előtt kell megkapni a fókuszt, amikor a felhasználó megnyomja a <tab> billentyűt.

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

5.5. Mnemonikus billentyűk

Egy konténer (**wx.Panel** és **wx.Dialog**) belsejében definiálni lehet (egy gomb számára) egy alapértelmezett parancsot, aminek a végrehajtására akkor kerül sor, amikor megnyomjuk az <Enter> billentyűt. Ehhez egyszerűen csak hívni kell a widget **SetDefault()** metódusát, amit ennek a funkciónak a megvalósítására szántak.

A programpéldában :

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

A különböző menüpontokhoz és (**kizárólag Windows alatt**) egy frame-ben lévő gombokhoz is lehet mnemonikus billentyűket asszociálni.

Ehhez egy három elemű tuple-kből álló listát kell létrehozni, ami az asszociációs tábla hivatalát fogja betölteni. Ezek a tuple-k lesznek polymorfizmus révén a **wx.AcceleratorEntry** -k és `(flag, keycode, cmdID)` formában jelennek meg.

A `flag` azokat a speciális billentyűket reprezentálja (<alt>, <ctrl>, etc …), amiket a felhasználónak le kell nyomni. A `wx.ACCEL_ALT, wx.ACCEL_SHIFT, wx.ACCEL_CTRL` konstansok definiálják őket és az 5.4 fejezetben látott bitmaszktechnikával kombinálhatók.

A `keycode` az a billentyű, amit a speciális billentyűkkel kell kombinálni, hogy aktiválódjon a parancs. Egy egész számról van szó, amit az **ord()** függvény alkalmazásával határozhatunk meg úgy, paraméterként a kívánt betűt adjuk meg.

Végül a annak a menüpontnak vagy annak a gombnak az id-je, amelyiket a mnemonikus billentyűvel kell asszociálni. Ez az első effektív alkalmazása azoknak az egyedi azonosítóknak, amikről az 5.4 fejezetben röviden szóltunk.

A példaprogram tartalmaz egy definíciós tábla példát gombok számára :

```
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())]
```

Ha már definiálva van a tábla, elegendő generálni egy **wx.AcceleratorTable** objektumot annak a frame-nek a **SetAcceleratorTable()** metódusával, amelyik frame a definiált billentyűk révén érintett lesz. Paraméterként meg kell adni a definíciós táblát.


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

Ahhoz, hogy a vezérlőcímkében alá legyen húzva a betű, egy "&" -t kell elé írni. pl.: "&File".

Mélyebb ismeretek szerzéséhez, illetve a rendszer tökéletesítéséhez a wxPython wiki egyik cikke javasol egy navigációs megoldást mnemonikus billentyűvel: (http://wiki.wxpython.org/index.cgi/FocusingControlsWithTheirLabels).

6. Főwidget-ek

Azután, hogy bemutattuk a gestion_liste.py példaprogramban a fókuszkezelés aspektusát, összefoglaló jelleggel áttekintjük a példaprogramot alkotó különböző widget-eket.

Ebben a fejezetben csak az alapwidget-eket és a velük asszociált főfunkciókat fogjuk bemutatni. A widget-ek lehetőségei gyakran sokkal nagyobbak, így ezen a szinten a wxPython wiki és a hivatalos dokumentáció hasznosak lesznek.

6.1. wx.StaticText

Más felhasználói interface könyvtárakban ezt a widget-et gyakran "label"-nek hívják. Felhasználó által nem módosítható egyszerű szöveg képernyőre történő kiíratását teszi lehetővé.

Konstruktora a következő alakú :

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

Semmi túlzottan új nincs a **wx.Window** konstruktorához képpes, csak a `label` paraméter megjelenése nem az. A paraméterben adjuk meg azt a szöveget, amit a generált widget ki fog írni.

A widget szövegét a **wx.Control** class-tól örökölt metódus segítségével lehet módosítani :

```
SetLabel(text)
```

Analóg módon, a szöveg a **GetLabel()** metódussal nyerhető ki.

Úgy akadályozhatjuk meg, hogy a widget szélessége túlságosan megnőjön, hogy n pixel után automatikus return-ök beszúrását kérjük a sorokba. Ez a következő metódus segítségével történik :

```
Wrap(width)
```

Figyelem : a widget szélességének levágása csak a szavak végén történik meg. Ha az engedélyezett méret kisebb, mint egyes szavak hossza, az problémákat okozhat.

6.2. wx.TextCtrl

Számos felhasználói interface készítésére szánt könyvtártól eltérően a wxPython-ban az egysoros és többsoros szövegbeviteli mezőt ugyanaz a widget reprezentálja.

Konstruktora :

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

A **wx.StaticText** `label` paramétere itt átalakult `value` -vá, de ugyanazt a funkciót reprezentálja.

A `validator` paraméter lehetővé teszi egy **wx.PyValidator** class-ból leszármaztatott objektum asszociálását. A validátor egy class, ami a **Validate()** metódus overload-olásával lehetővé teszi egy kontrol tartalmának egyszerű vizsgálatát. Ebben a dokumentumban nem fogjuk részletezni ezt a lehetőséget.

Egy **wx.TextCtrl** -nak alapértelmezetten csak egyetlen szövegsora lehet. Ez megváltoztatható a

`wx.TE_MULTILINE` stílus alkalmazásával.

A felhasználó által beírt szöveg kinyeréséhez több metódus is rendelkezésünkre áll (mindegyik visszatérési értéke egy `String`)

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

Az elsővel egy adott sor szövegét nyerhetjük ki a sor sorszámának megadásával. Mint a programpéldában, a **`wx.TextCtrl`** egysoros, ezzel a metódushívással az egész szöveg kinyerhető :

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

A második és a harmadik metódussal két karakterpozíció között lehet kinyerni a szöveget. (Az első karakter a 0 pozícióban van.) Egy többsoros **`wx.TextCtrl`** esetében egy egyszerű módszer az egész szöveg kinyerésére :

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

Végül a negyedik metódussal a felhasználó által kiválasztott szöveget nyerhetjük ki a widget-ből.
A szöveget programozott módon is be lehet írni a következő metódusoknak köszönhetően :

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

Az első a widget-beli régi szöveget helyettesíti a `value` karakterlánccal.

A második viselkedése megegyezik az elsőével, csak itt egy szövegfile tartalmával helyettesíti a widget-beli régi szöveget.

A két következő metódus a widget-ben már meglévő szöveg végéhez, illetve a beszúrókurzor után illeszti a szöveget.

Mindegyik esetben a szöveghozzáadó metódus végrehajtásának végén a beszúrókurzor a beszúrt szöveg végén található.
Szükség esetén lehetőség van a beszúrókurzor manipulálására a **`GetInsertionPoint()`** és a **`SetInsertionPoint()`** metódusok segítségével. Használatuk módját nem nehéz kitalálni.

Végezetül, lehetőség van a szövegstílus globális vagy a **`wx.TextCtrl`** -re lokalizált paraméterezésére A ehhez metódusok mint egy **`wx.TextAttr`** class állnak rendelkezésünkre. Az olvasóra hagyjuk a wxPython és wxWidgets dokumentációk kínálta lehetőségek felfedezését.

6.3. wx.Button

A menük és a mnemonikus billentyűk mellett a gombok a felhasználó főeszközei, amik az alkalmazásbeli kívánságai érvényesítésére szolgálnak.

A szöveget tartalmazó standard gombok generálására való **`wx.Button`** class mellet a wxPythonnak van egy másik **`wx.BitmapButton`** subclass-a, ami lehetővé teszi a gombban a szöveg képpel történő helyettesítését.

A viszonylag fejlett alkalmazásokban a gombok általában az eszközsávban vannak csoportosítva,

amire vonatkozóan a referencia kézikönyvben a **wx.ToolBarBase** class és leszármazottainak leírását tanulmányozva szerezhet az olvasó információkat.

A **wx.Button** class konstruktora a következő :

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

A class-nak nincs sok metódusa. Tekintettel arra, hogy a **wx.Control** leszármazottja, a **wx.StaticText** -nél látott **SetLabel()** és **GetLabel()** metódusok teszik lehetővé a gomb szövegének manipulálását.

6.4. wx.ListBox

Ez a class egy választható karakterláncokból álló listát reprezentál. Az ilyen listákat általában arra használják, hogy a felhasználóknak egyszerre több kiválasztást engednek meg a programban. A példaprogramban adattárolóként használjuk, amit könnyű lesz szinkronizálni például egy kis adatbázissal.

Először is tudni kell, hogy a **wx.ListBox** egy **wx.ItemContainer** interface-ből származik, ami rendelkezésünkre bocsátja azokat az elem manipulációs metódusokat, amiket a **wx.ComboBox** -okban találnánk. Melegen ajánljuk ez utóbbiak tanulmányozását, ezeket nem fogjuk tárgyalni ebben a jegyzetben.

A konstruktor a következő alakú :

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

A `choices` paraméter egy lista vagy egy tuple, ami karakterláncokból áll, melyek utóbbiak egyesével lesznek beolvasva és beletéve a **wx.ListBox** -ba.

Számításba kell venni a `style` paramétert is. A **wx.TextCtrl** mintájának megfelelően egy **wx.ListBox** két -féle módon működhet. Vagy csak egy elem kiválasztását fogadja el (`wx.LB_SINGLE`), vagy egyidejűleg több elem is kiválasztható.

Ráadásul az utóbbi működési mód még két féle képpen viselkedhet. Vagy az történik, hogy egy elemre történő kattintás kiválasztja az illető elemet, illetve megszünteti az illető elem kiválasztását (`wx.LB_MULTIPLE`), vagy a felhasználó használhatja a <shift> és <ctrl> billentyűket több elem kiválasztására, illetve több elem kiválasztásának megszüntetésére (`wx.LB_EXTENDED`).

Alapértelmezetten egy **wx.ListBox** stílusa `wx.LB_SINGLE`.

Egy **wx.ListBox** -nak számos olyan metódusa van, amik hasonlítanak egy Python lista azon metódusaira, amik egy listaelem hozzáadására, módosítására vagy törlésére szolgálnak az elem indexe alapján. Ennek ellenére ez a widget-et nem feltétlenül erre a fajta használatra készítették. A következő metódus való a tartalma generálására :

```
Set( items )
```

Ez a metódus módosítja a konstruktorban látott `choice` paramétert. Ez a művelet megsemmisíti a lista régi tartalmát.

Egy **wx.ListBox** -nak az az értelme, hogy választási lehetőséget kínál a felhasználónak, ezért képeseknek kell lennünk arra, hogy megtudjuk, mit választott a felhasználó.

Egy elem kiválasztását megengedő **wx.ListBox** esetében (mint a milyen a példaprogramban is van) a következő metódusok segítségével tudhatjuk meg az indexet illetve a kiválasztott objektum tartalmát :

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

Ha semelyik elemet sem választottuk ki, akkor az első metódus visszatérési értéke a wx.NOT_FOUND konstans, míg a másodiké egy üres karakterlánc lesz. Észre fogják venni, hogy ezek ugyanazok a metódusok, mint amiket a "Párbeszéddobozok többszörös kiválasztással" című fejezetben láttunk.

Többszörös választást megengedő **wx.ListBox** esetében egyetlen egy metódus áll a rendelkezésünkre :

```
GetSelections()
```

A listából kiválasztott elemek indexeiből álló tuple-t ad visszatérési értékként.

Végezetül, a **wx.ListBox** -hoz két nagyon hasznos esemény tartozik a : wx.EVT_LISTBOX és a wx.EVT_LISTBOXDCICK. Az előbbi akkor generálódik, amikor a felhasználó kiválaszt egy elemet, a második akkor, amikor a felhasználó kettőt kattint egy listaelemre.

A példaprogramban a "Modifier" és "Supprimer" gombok csak akkor aktiválódnak, amikor a felhasználó kiválasztottunk egy listaelemet.

```
GetSelections()
```

Jó gyakorlat lenne a program olyan tökéletesítse, hogy a felhasználónak azonnal ajánljuk fel annak az elemnek a módosítását, amire duplán kattintott.

6.5. wx.RadioBox/wx.CheckListBox

Ez a két widget meglehetősen hasonlít egymásra, lehetővé teszik, hogy a felhasználó több lehetőség közül : egyetlen egyet válasszon ki a **wx.RadioBox** esetében, illetve hogy egyidejűleg többet választhasson ki a **wx.CheckListBox** esetében.

A *radio_check.py* programban láthatjuk, hogy hogyan jelennek meg ezek a widget-ek egy grafikus interface-ben.

Egy **wx.RadioBox** -ban tett kiválasztásokat "rádiógomboknak" (radio button) hívjuk. A **wx.RadioButton** class teszi lehetővé ezek manuális definiálását. A **wx.RadioBox** egy egyszerű és kényelmes módot kínál az utóbbiak csoportosítására.

Egy wx.RadioBox konstruktor a következő alakú :

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

A label paraméter egy String és a rádiógombok csoportjához asszociált szöveget reprezentálja. A programpéldában látjuk, hogy a keretbe íródik ki.

A choices paraméter egy stringekből álló lista vagy tuple, ami a felhasználó rendelkezésére álló különböző választási lehetőségeket reprezentálja.

A majorDimension paraméter meghatározza a stílusválasztástól függően a widget maximális sor

vagy oszlopszámát.

A stílus a **wx.Window** -tól örökölt értékeken felül fölveheti a wx.RA_SPECIFY_COLS illetve wx.RA_SPECIFY_ROWS értékeket. Ezek az értékek teszik lehetővé annak meghatározását, hogy a majorDimension paramétert oszlopra vagy sorra alkalmazzuk, amikor a widget majd elrendezi a választási lehetőségeket.

A programpéldában a stílus paraméternek a wx.RA_SPECIFY_ROWS és a majorDimension paraméternek a 0 (semleges) értéket állítottuk be, tehát a 4 választási lehetőség különböző sorokba íródik ki. Ha például a majorDimension értékét 2-re állítottuk volna be, akkor a választási lehetőségek 2 oszlopba íródtak volna ki, mert a widget ebben az esetben azt írtuk volna elő, hogy csak két sort tartalmazhat.

Az előző részben látott **wx.ListBox** -hoz képpest az a különbség, hogy a **wx.RadioBox** -nak semmilyen metódusa sincs a rendelkezésre álló választási lehetőségek megváltoztatására. Ez egy statikus struktúra a felhasználói interface-ben.

A felhasználó választását ugyanazokkal a metódusokkal tudhatjuk meg, mint amikkel egy **wx.ListBox** esetében :

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

Ezek a metódusok ugyanúgy működnek, mint egy **wx.ListBox** azonos nevű metódusai és mint egy **wx.ListBox**, a **wx.RadioBox** mindig generál egy wx.EVT_RADIOBOX eseményt, amikor a felhasználó rákattint egy rádiógombra.

A **wx.CheckListBox** még közelebb áll egy **wx.ListBox** -hoz, mivel közvetlenül abból származik !

Ebből a tényből eredően a megjelenése eltér a szokásos grafikus interface-ekben megszokott jelölőnégyzet megjelenéstől.

Egy **wx.CheckListBox** konstroktorának a következő a formája :

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

Ez a konstruktor pontosan ugyanolyan, mint a **wx.ListBox** konstruktora. A **wx.CheckListBox** két metódust és egy kiegészítő eseményt bocsájt a rendelkezésünkre :

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

Az első lehetővé teszi, hogy egy jelölő négyzetet az indexe alapján bejelöljünk (vagy a bejelölést megszüntessük, a check paraméternek a False értéket adva).

A második lehetővé teszi – ugyancsak az indexe alapján – , hogy megtudjuk, a jelölő négyzet be van-e jelölve vagy sem.

Nagyon fontos, hogy különbséget tegyünk egy kiválasztott elem és egy bejelölt között. Az örökölt **GetSelections()** metódus nem fogja visszatérési értékként megadni a bejelölt jelölőnégyzetek indexlistáját, hanem helyesen, valóban a a kiírt listában standard módon kiválasztott elemek indexeinek a listáját.

Így a felhasználó kiválasztásainak megismeréséhez végig kell iterrálnunk az összes indexen és minden iterrációs lépésnél ellenőriznünk kell, hogy a kontrol be van-e jelölve, mielőtt valamilyen akciót hajtanánk végre.

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

A **wx.CheckListBox** egy új eseményt - ez a wx.EVT_CHECKLISTBOX esemény – biztosít a programozónak annak érdekében, hogy különbséget lehessen tenni a kiválasztás/kiválasztás törlése és a bejelölés/ bejelölés törlése között.

Az olvasó minden bizonnyal észrevette, hogy ez a widget nem igazán konvencionális. A felhasználó általában hozzá van szokva, hogy jelölőnégyzetekből álló felületei vannak, amiknek a megjelenése hasonlít egy **wx.RadioBox** -ra. Ilyen eredmény úgy kapható, hogy egy **wx.RadioBox** -szal összekapcsolt **wx.StaticBoxSizer** belsejében egymásbarakunk egy **wx.Panel** -t és **wx.CheckBox** widget-eket. Jó gyakorlat lesz ezt az utat követve kifejleszteni egy **wx.RadioBox** -ra hasonlító **wx.CheckBoxBox** class -t.

7. Grafikus elemek manipulálása

A wxPython absztrakciós rétegeket használ a grafikus elemek manipulálására. A legprimerebb absztrakciós réteget Device Context-nek nevezik. Egy class-ról van szó, ami metódusokat szolgáltat olyan perifériákra való rajzoláshoz, mint amilyen a képernyő, a nyomtató, stb.

7.1. Hogyan töltünk be egy képet

A wxPythonos képmanipulációk két class-t hívnak, ezek a **wx.Image** és **wx.Bitmap**. Az első platformfüggetlen, míg a második platformfüggő. A gyakorlatban ez azt jelenti, hogy a manipulációk a **wx.Image** segítségével történnek és a **wx.Bitmap** a képeknek a képernyőre történő kiírásával foglalkozik.

Egy kép betölthető a **wx.Image** konstruktorának hívásával :

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

A `name` paraméter a betöltendő képfile neve, a `type` paraméter a kép típusa (lásd az alábbi táblázatot) és ha a fájl több képet tartalmaz (ez lehet a helyzet például a GIF file-ok esetében), az `index` paraméterrel lehet meghatározni a képet.

A következő, koránt sem kimerítő táblázat néhány támogatott képformátumot sorol fel :

type (flag)	Leírás
wx.BITMAP_TYPE_JPEG	JPEG típusú kép
wx.BITMAP_TYPE_GIF	GIF típusú kép
wx.BITMAP_TYPE_PNG	PNG típusú kép
wx.BITMAP_TYPE_ICO	Windows ikonok formátuma
wx.BITMAP_TYPE_TIF	TIFF típusú kép
wx.BITMAP_TYPE_ANY	A wxPython megkísérli automatikusan detektálni a kép típusát

7.2. Képmanipuláció

A **wx.Image** class különböző képműveleteket kínál. A következő táblázat a leggyakrabban végzett műveleteket sorolja fel :

Metódus	Leírás
Rescale(width, height)	A width és height új értékével változtatja meg a kép méretét.
Rotate(angle, **rotationCentre,** **interpolating=True,** **offsetAfterRotation=None)**	A rotationCentre centrumhoz (egy **wx.Point**) képpest angle radián szöggel végrehajtott forgatás után egy új **wx.Image** objektum a visszatérési értéke. Egy hosszabb, de pontosabb algoritmust használ, ha az interpolating értéke True.
Rotate90(clockwise=True)	90 -kal elforgatja a képet. Ha a paraméter értéke True, akkor az óramutató járásának megfelelő irányba forgat.
Scale(width, height)	Visszatérési értéke a kép másolata, miután width és height méretűre változtatta a kép méreteit.
Mirror(horizontally=True)	Egy wx.Image objektum a visszatérési értéke. A kép a vízszintes tengelyre van tükrözve (ha a paraméter értéke True), vagy a függőleges tengelyre, (ha a paraméter értéke False). Az eredeti képet nem módosítja.
Replace(r1, g1, b1, r2, g2, b2)	Mindegyik pixelt, amelyiknek a színe (r1, g1, b1) az (r2, g2, b2) színnel helyettesíti.

Tudjon róla az olvasó, hogy egy maszkot alkalmazva (**SetMaskColor()** metódus) játszani lehet a képek átlátszóságával, vagy játszani lehet a pixelek alfa-értékével.

Itt nem fogunk belemenni ebbe a témába, viszont az olvasó további információkat fog találni a témáról a wxPython wiki-ben (http://wiki.wxpython.org/index.cgi/WorkingWithImages) valamint az API dokumentációjában (http://www.wxpython.org/docs/api/frames.html).

8. Menük

A wxPython a a menüket három class segítségével kezeli :

- **wx.MenuBar** : magának a menüsávnak a kezelésével foglalkozik
- **wx.Menu** : az egyes menük (pl.: File menü) vagy almenük kezelésére való
- **wx.MenuItem** : egy menüpontot reprezentál

A menügenerálás hét lépésben foglalható össze :

1. A menüsáv létrehozása
2. A menüsáv hozzákapcsolása egy frame-hez
3. Az egyes menük létrehozása
4. Ezeknek a menüknek a hozzákapcsolása a menüsávhoz vagy egy szülőmenühöz
5. A menüelemek (menüpontok) létrehozása
6. A menüelemek hozzákapcsolása a megfelelő menühöz
7. Minden menüelemmel (menüponttal) összekötünk egy akciót

Az előbb leírt lépések végrehajtásának sorrendje rugalmas és változhat. Minden esetre egy menü kreálása során automatikusan végigmegyünk a fönti 7 lépésből álló műveletsoron.

8.1. Menüsáv létrehozása

Egy menüsáv generálása a **wx.MenuBar** konstruktorával történik, aminek nincs paramétere.

A menüsáv egy frame-hez történő kapcsolásához a **wx.Frame SetMenuBar(menubar)** metódusát kell hívni.

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

8.2. Menük létrehozása

A legördülő menük generálása a **wx.Menu()** konstruktorral történik :

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

A menük számára csak egy stílus lehetséges és azt csak a GTK támogatja. A `wx.MENU_TEAROFF` stílusról van szó, ami lehetővé teszi a menü leválasztását a menüsávról annak érdekében, hogy az előbbi autonóm módon legyen használható. A `title` paraméter titok marad, nem fogjuk használni a példáinkban.

8.3. Egy menü hozzákapcsolása/törlése a menüsorhoz/ból

A wx.MenuBar -nak három – a menüsáv menüinek kezelésére való – érdekes metódusa van.

Metódus	Leírás
Append(menu, title)	A paraméterben átadott menüt hozzáadja a menüsávhoz és egy közös nevet asszociál hozzá.
Insert(pos, menu, title)	A paraméterben átadott menüt beszúrja a menüsávba a pos pozícióba.
Remove(pos)	Törli a pos pozícióban lévő menüt.

8.4. Menüpontok hozzáadása a menühöz

A menüpontok hozzáadásának vonatkozásában különböző megközelítések vannak (például a Quit menüpont hozzáadása a File menühöz).

A legtömörebb módszer a **wx.Menu Append()** metódusának alkalmazása :

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

Az `id` paraméter egy wxPython ID. A `text` paraméter egy karakterlánc, ami majd ki lesz írva a menüben (mint a menüpont neve). Ha a `helpStr` argumentum definiálva van és a frame tartalmaz egy státussávot, akkor a `helpStr` a státussávban fog kiíródni amikor a menüpontot kiemeljük. Ez a metódus a menü végéhez ad egy menüpontot egy implicit **wx.MenuItem** objektum generálásával.

A **wx.Menu AppendSeparator()** metódusával elválasztó szakaszokat adhatunk a menühöz.

Alkalmazási példa :

```
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'
```

A menüpontok hozzáadásának egy másik megközelítése az, hogy a menüpontokat a (**wx.MenuItem()** konstruktorral) explicit módon hozzuk létre, majd hozzákapcsoljuk a menühöz. A menüpontokat generáló konstruktor a következő alakú :

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

A `parentMenu` paraméter a menüpont szülőmenüjét reprezentálja és a **wx.Menu** class egy objektumának kell lennie. A szülő megadásának ténye nem fogja automatikusan előidézni a menüpont menühöz való hozzákapcsolását. Ezt az **AppendItem()** metódussal explicit módon kell megtenni ! A `subMenu` paraméter megadásával lehetőségünk van ennek a menüpontnak almenüvé való átalakítására. Mi minden esetre azt tanácsoljuk az olvasónak, hogy ne használja ezt a módszert,

hanem alkalmazza ennek a fejezetnek az "Almenük" alfejezetében bemutatott megközelítést.

Egy másik módszer ugyanennek az eredménynek az eléréséhez :

```
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. Menüesemény kezelése

Egy akciónak egy menüponthoz kapcsolása emlékeztet arra, ahogy egy gombhoz kapcsolunk akciót. Az elv ugyanaz, a **`Bind()`** metódust kell alkalmazni a menüsávot tartalmazó frame-re a `wx.EVT_MENU` eseménnyel.

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

Ahol : `self` a frame, `self.OnExit` a hívandó metódus és `itemQuitter` a menüpont.

8.6. Gyorsítóbillentűk (short cut) és mnemonikus hozzáférés

Mnemonikus billentyűk kapcsolhatók a menüpontokhoz úgy, hogy a kiírandó szöveg általunk kiválasztott betűje elé egy **&** jelet írunk. Például ahhoz, hogy az "`Enregistrer sous...`" 'n'-jéhez mnemonikus hozzáférést kapcsoljunk, "E`&`nregistrer sous..." -t kell írnunk.

Ugyanígy, létezik egy egyszerű módszer arra, hogy egy menüponthoz gyorsítóbillentyűt kapcsoljunk. Ehhez elég belevenni a billentyű kombinációt a menüpont szövegébe. A szöveg végéhez hoozá kell írni a `\t` tabulációs karaktert és ezt kell követni a billentyű kombinációnak. Ez a kombináció kezdődhet a `Ctrl`, `Alt` vagy `Shift` módosítóbillentyűkkel, utána jöhet a '+' vagy '-' elválasztó majd a kívánt karakter következik.

Egy példa világosabbá fogja tenni az elmondottakat : a `Ctrl+S` gyorsítóbillentyűt akarjuk hozzárendelni az "`Enregistrer`" menüponthoz, tehát elegendő "`Enregistrer\tCtrl+S`" (vagy "`Enregistrer\tCtrl-S`") -t írni. A módosítóbillentyűket kombinálni lehet. Pl.: "`Enregistrer Sous\tCtrl+Shit+S`".

MacOS X alatt a `Ctrl` módosítóbillentyű a Command (Apple) billentyűt reprezentálja.

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

8.7. Almenük

Almenük hozzáadásához a **wx.Menu** class az AppendMenu() metódust kínálja. Prototípusa :

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

Tekintsünk egy példát :

```
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. Következtetés

Befejezésül : reméljük az olvasónak sikerült általános rálátást szerezni a wxPythonra és funkcióira.

Tudatában vagyunk annak, hogy sok mindennel nem foglalkoztunk. A wxPythonnak van angol nyelvű bibliográfiája, továbbá rendelkezésünkre áll a "wxPython in action" [1] című kiváló könyv.

Ráadásul a wxPython közösség nagyon aktív és a wikiben számos komplett és magyarázatokkal jól ellátott példát találunk. A közösség eszközöket is fejleszt : mint az interface-fejlesztő, ami a fejlesztés végső stádiumában van. Bár nem beszéltünk róluk ebben a jegyzetben, de javasoljuk az olvasónak, hogy vessen egy pillantást a Glade és a BoaConstructor alkalmazásokra.

10. Bibliográfia

[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. A példák forráskódja

11.1. Hello World (hello.py)

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

class MainApp(wx.App):
    """ A wxPython által felkínált Application class-ból generáljuk az alkalmazást.
        Figyelem: A "constructor" egy kicsit speciális !"""
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Hello World !')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Egy - az alkalmazásunkra adaptált ablak generálása. A wxPython által
        felkínált alap ablak-class
        Ezért származtattuk le az ablakunkat a wxPython által felkínált alap ablak-
        class-ból.. """
    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. Példa egyszerű esemény kezelésére (goodbye.py)

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

class MainApp(wx.App):
    """ A wxPython által felkínált application class-ból leszármaztatott
        alkalmazás generálása.
        Figyelem: A "constructor" egy kicsit speciális !"""
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Hello World !')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Egy - az alkalmazásunkra adaptált ablak generálása. A wxPython által
        felkínált alap ablak-class
        Ezért származtattuk le az ablakunkat a wxPython által felkínált alap ablak
        -class-ból... """
    def __init__(self, title):
        """
            Itt változás van. A frame méretét és a stílusát adjuk meg.
            A kifejezés azt jelenti, hogy használjuk az alapértelmezett
            ablakstílust és abból vegyük ki (^ operátor) az ablak átméretezésének
            lehetőségét és a címsor három gombját.
            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) )
        # Egy widget-konténert teszunk a frame-be
        panel = wx.Panel(self, wx.ID_ANY)
        # Egy gombot teszunk a konténerbe egy kiválasztott pozícióba
        self.button = wx.Button(panel, wx.ID_ANY, "Good bye !", pos=(100, 30))
        # A button objektum EVT_BUTTON eseményéhez hozzárendeljuk az OnClick metódust
        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. Dialogbox-ok (dialog.py)

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

class MainApp(wx.App):
    """ A wxPython által felkínált application class-ból leszármaztatott
        alkalmazás generálása."""
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Párbeszéddoboz')
        self.SetTopWindow(frame)
        return True

    def OnExit(self):
        """ Az alkalmazásból történő kilépés előtt automatikusan hívott metódus"""

        # Egy egyszeru párbeszéddoboz objektum generálása
        dialogue = wx.MessageDialog(None, 'GoodBye!', "message d'au revoir",
                                    wx.OK)

        # a párbeszéddoboz kiíratása
        result = dialogue.ShowModal()
        return True

class MainWindow(wx.Frame):
    """ Egy - az alkalmazásunkra adaptált ablak generálása. A wxPython által
        felkínált alap ablak-class
        Ezért származtattuk le az ablakunkat a wxPython által felkínált alap ablak-
        class-ból..."""

    def __init__(self, title):
        # Itt változás van. A frame méretét adjuk meg.
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(300, 150))

        # Egy widget-konténert teszünk a frame-be
        self.panel = wx.Panel(self, wx.ID_ANY)

        # Sizer a gombok elhelyezéséhez
        sizer = wx.GridBagSizer(5,5)

        # Gombok elhelyezése a konténerbe
        btnDialog = wx.Button(self.panel, wx.ID_ANY, "Egyszerű párbeszéddoboz",
                              size=(290, 35))
        btnSaisie = wx.Button(self.panel, wx.ID_ANY, "Párbeszéddoboz adatbeírással",
                              size=(290, 35))
        btnChoix1 = wx.Button(self.panel, wx.ID_ANY, "Párbeszéddoboz egyszeres
                              választással",
                              size=(290, 35))
        btnChoix2 = wx.Button(self.panel, wx.ID_ANY, "Párbeszéddoboz többszörös
                              választással",
                              size=(290, 35))
        btnChoix3 = wx.Button(self.panel, wx.ID_ANY, "Egy szín kiválasztása",
                              size=(290, 35))
        btnExit    = wx.Button(self.panel, wx.ID_ANY, "&Kilépés")

        # Sizer hozzáadása
        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))   # alsó margó
        sizer.Add((10,10), pos=(0,2), span=(5, 1))   # jobboldali margó

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

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

        # Metódusok hozzárendelése a gombra történő kattintáshoz
        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):
        """ Egyszerű párbeszéddoboz kiíratása """
        dlgMsg = wx.MessageDialog(None, 'Szereti ezt a példát ?',
                                  'Kérdés',
                                  wx.YES_NO | wx.YES_DEFAULT | wx.ICON_QUESTION)
        reponse = dlgMsg.ShowModal()
        if reponse == wx.ID_YES:
            message = 'Klassz !'
        else:
            message = 'Oh, nem!'
        wx.MessageDialog(self, message, style=wx.OK).ShowModal()
        dlgMsg.Destroy()

    def AfficherDlgSaisie(self, source):
        """Adatbeírásos párbeszéddoboz kiíratása """
        dlgSaisie = wx.TextEntryDialog(self, 'Írja be a keresztnevét',
                                             'Személyes kérdés',
                                             'ide írja be a szöveget',
                                              style=wx.OK|wx.CANCEL)
        # Párbeszéddoboz kiíratása + a visszatérési érték kinyerése
        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):
        """ Egyszeres kiválasztású párbeszéddoboz kiíratása """
        dlgMulti = wx.SingleChoiceDialog(self,
                    'Melyik a kedvenc gyümölcse ?',
                    'Egyszeres választás',
                    [u'őszibarack', u'alma', u'sárgabarack', u'gesztenye',
                     u'szilva'])
        if dlgMulti.ShowModal() == wx.ID_OK:
            fruit = dlgMulti.GetStringSelection()
            wx.MessageDialog(None,
```

```
                        u'Egy ' + fruit + u'torta készül', 'Mi készül',
                        style=wx.OK).ShowModal()
        dlgMulti.Destroy()

    def AfficherDlgChoixMulti(self, source):
        """ Többszörös kiválasztású párbeszéddoboz kiíratása """
        fruits = [u'őszibarack', u'alma', u'sárgabarack', u'gesztenye', u'szilva']
        dlgMulti = wx.MultiChoiceDialog(self,
                    'Melyek a kedvenc gyümölcsei ?',
                    'Többszörös választás', fruits)

        if dlgMulti.ShowModal() == wx.ID_OK:
            # a kiválasztott elemek kinyerése egy indexlista formájában
            listeFruits = u""
            for i in dlgMulti.GetSelections():
                listeFruits = listeFruits + fruits[i] + u", "
            wx.MessageDialog(None,
                    u"Éppen egy " + listeFruits + u" torta készül...",
                    style=wx.OK).ShowModal()
        dlgMulti.Destroy()

    def AfficherDlgCouleur(self, source):
        """ A ColourDialog kiíratása """
        dlgCouleur = wx.ColourDialog(self)
        res = dlgCouleur.ShowModal()
        if res == wx.ID_OK:
            # a DataColour kinyerése
            couleur = dlgCouleur.GetColourData()
            # a panel hátterének színezése a kiválasztott színnel
            self.panel.SetBackgroundColour(couleur.GetColour())
        dlgCouleur.Destroy()

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

11.4. Példa a GridSizer használatára (gridsizer.py)

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

# A gombok neveinek listája
labels = "un deux trois quatre cinq six sept huit neuf".split()

class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('GridSizer')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """Ablak a Grid sizer bemutatásához. """

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

        # Egy panel létrehozása
        self.panel = wx.Panel(self);

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

        # Gombok létrehozása és hozzáadása a sizer-hez
        for label in labels:
            # Ebben a példában semmilyen akciót sem rendelünk a gombokhoz.
            btn = wx.Button(self.panel, wx.ID_ANY, label);
            self.sizer.Add(btn, 0, 0)

        # A sizer-t a panelhez kapcsoljuk
        self.panel.SetSizer(self.sizer)
        # Beállítja a sizer méretét
        self.sizer.Fit(self)
        # Az ablak minimális méretének definiálása
        self.SetMinSize((300, 150))

        # Az ablak megjelenítése
        self.Show(True)

if __name__ == "__main__":
    app = MainApp(redirect=False) # Arra való, hogy a standard kimeneti stream-et
                                  # ne a grafikus ablakba irányítsuk
                                  # (debug-oláshoz hasznos)
    app.MainLoop()
```

11.5. Példa a FlexGridSizer használatára (flexgridsizer.py)

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

# A gombok neveinek listája
labels   = "un deux trois quatre cinq six sept huit neuf dix onze douze".split()
couleurs = ["red", "gray", "sky blue"]

class MainWindow(wx.Frame):
    """Ablak a Flex Grid sizer bemutatásához. """

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

        # A grid sizer létrehozása
        sizer = wx.FlexGridSizer(rows=4, cols=3, hgap=2, vgap=2)
        i=0
        # panelek létrehozása és hozzáadása a sizer-hez
        for label in labels:
            panel = PanelColore(self, wx.ID_ANY, label, couleurs[i%len(couleurs)]);
            sizer.Add(panel, 0, wx.EXPAND) # wx.EXPAND lehetővé mteszi a panel
                                           # méretének változtatását
            i=i+1

        sizer.AddGrowableCol(2) # a 3. oszlopot méretváltoztatóvá
        sizer.AddGrowableRow(1) # a 2. sort méretváltoztatóvá teszi
        sizer.AddGrowableRow(3) # a 4. sort méretváltoztatóvá teszi

        # a sizer-t a frame-hez kapcsolja
        self.SetSizer(sizer)
        # beállítja a sizer méretét
        self.Fit()
        # az ablak minimális méretének definiálása
        self.SetMinSize((300, 120))
        # az ablak megjelenítése
        self.Show(True)

class PanelColore (wx.Panel):
    """ Panel háttérszínnel annak érdekében, hogy jól
        megkülönböztethessük a sizer-ek különböző beállításait." ""

    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() # panel méretének kinyerése
        dc = wx.PaintDC(self) # a grafikus kontextus kinyerése
                              # (DC = Device Context)
        # a szélesség és a magasság kinyerése, amit a kiírandó szöveg fog felvenni
        w,h = dc.GetTextExtent(self.label)
        # a szöveg kirajzolása a panelbe
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)
```

```
class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('FlexGridSizer')
        self.SetTopWindow(frame)
        return True

if __name__ == "__main__":
    app = MainApp(redirect=False) # Arra való, hogy a standard kimeneti stream-et
                                  # ne a grafikus ablakba irányítsuk
                                  # (debug-oláshoz hasznos)
    app.MainLoop()
```

11.6. Példa a GridBagSizer használatára (gridbagsizer.py)

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

# A gombok neveinek listája
couleurs = ["red", "gray"]
# A gombok neveinek listája
labels   = "un deux trois quatre cinq six".split()

class MainWindow(wx.Frame):
    """Ablak a Grid Bag sizer bemutatásához. """

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

        # A grid bag sizer létrehozása
        sizer = wx.GridBagSizer(hgap=2, vgap=2)

        # 6 panel létrehozása és a sizer-hez kapcsolásuk
        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))

        # Annak a panelnek a hozzáadása, ami az egész 3. oszlopra lesz pozícionálva
        panel = PanelColore(self, wx.ID_ANY, "4 sor egyesítése", "sky blue");
        sizer.Add(panel, pos=(0, 2), span=(4,1), flag=wx.EXPAND)

        # Annak a panelnek a hozzáadása, ami az első sor első 2 oszlopára
        # lesz pozícionálva
        panel = PanelColore(self, wx.ID_ANY, "2 oszlop egyesítése", "gold");
        sizer.Add(panel, pos=(3, 0), span=(1,2), flag=wx.EXPAND)
        sizer.AddGrowableCol(2) # a 3. oszlopot méretváltoztatóvá teszi
        sizer.AddGrowableRow(3) # a 4. sort méretváltoztatóvá teszi

        # a sizer-t a frame-hez kapcsolja
        self.SetSizer(sizer)

        # beállítja a sizer méretét
        sizer.Fit(self)

        # az ablak megjelenítése
        self.Show(True)

class PanelColore (wx.Panel):
    """ Panel háttérszínnel annak érdekében, hogy jól
        megkülönböztethessük a sizer-ek különböző beállításait.
    """
    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() # panel méretének kinyerése
        dc = wx.PaintDC(self) # a grafikus kontextus kinyerése
                              # (DC = Device Context)
        # a szélesség és a magasság kinyerése, amit a kiírandó szöveg fog felvenni
        w,h = dc.GetTextExtent(self.label)
        # a szöveg kirajzolása a panel közepére
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)

class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Grid Bag Sizer')
        self.SetTopWindow(frame)
        return True

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

11.7. Példa a BoxSizer használatára (boxsizer.py)

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

class MainWindow(wx.Frame):
    """Ablak a Box sizer bemutatásához. """

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

        # A grid sizer létrehozása
        sizer = wx.BoxSizer(wx.VERTICAL)

        # A panelek definíciója
        haut    = PanelColore(self, wx.ID_ANY, "Felső panel", "red");
        milieu = PanelColore(self, wx.ID_ANY, "Középső panel", "coral");
        bas     = PanelColore(self, wx.ID_ANY, "Alsó panel", "lime green");

        # Gombok hozzáadása a sizer-hez
        sizer.Add(haut,    proportion=0, flag=wx.EXPAND)
        """ Az arány 1 azért, hogy  ez a widget elfoglalja az egész üres területet"""
        sizer.Add(milieu, proportion=1, flag=wx.EXPAND)
        sizer.Add(bas,     proportion=0, flag=wx.EXPAND)

        # a sizer-t a frame-hez kapcsolja
        self.SetSizer(sizer)

        # beállítja a sizer méretét
        self.Fit()

        # az ablak megjelenítése
        self.Show(True)

class PanelColore (wx.Panel):
    """ Panel háttérszínnel annak érdekében, hogy jól
        megkülönböztethessük a sizer-ek különböző beállításait.
    """
    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() # panel méretének kinyerése
        dc = wx.PaintDC(self) # a grafikus kontextus kinyerése (DC = Device
                              # Context)
        # a szélesség és a magasság kinyerése, amit a kiírandó szöveg fog felvenni
        w,h = dc.GetTextExtent(self.label)
        # a szöveg kirajzolása a panel közepére
        dc.DrawText(self.label, (size.width-w)/2, (size.height-h)/2)
```

```
class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('GridSizer')
        self.SetTopWindow(frame)
        return True

if __name__ == "__main__":
    app = MainApp(redirect=False) # Arra való, hogy a standard kimeneti stream-et
                                  # ne a grafikus ablakba irányítsuk
                                  # (debug-oláshoz hasznos)
    app.MainLoop()
```

11.8. Példa a StaticBoxSizer használatára (staticboxsizer.py)

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

class MainWindow(wx.Frame):
    """Ablak a Static Box sizer bemutatásához. """

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

        # Egy panel létrehozása
        self.panel = wx.Panel(self)

        # Egy static box és a vele járó sizer létrehozása
        groupe1 = wx.StaticBox(self.panel, wx.ID_ANY, "Groupe 1")
        sizerGrp1 = wx.StaticBoxSizer(groupe1, wx.VERTICAL)

        # Ugyanez a művelet: a második static box és a vele járó sizer létrehozása
        groupe2 = wx.StaticBox(self.panel, wx.ID_ANY, "Groupe 2")
        sizerGrp2 = wx.StaticBoxSizer(groupe2, wx.VERTICAL)

        # A gombok definíciója és hozzáadása a 2 static box sizer-hez
        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)

        # A befoglaló sizer létrehozása a 2 static box pozícionálásához
        sizer = wx.BoxSizer(wx.HORIZONTAL)
        sizer.Add(sizerGrp1, flag=wx.ALL, border=10)
        sizer.Add(sizerGrp2, flag=wx.ALL, border=10)

        # a sizer-t a panel-hez kapcsolja
        self.panel.SetSizer(sizer)

        # beállítja a sizer méretét
        sizer.Fit(self)

        # az ablak megjelenítése
        self.Show(True)

class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Static Box Sizer')
        self.SetTopWindow(frame)
        return True

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


11.9. Alkalmazási példa (gestion_liste.py)

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

class ListControler:
    """ Kísérlet egy minimális grafikus interface-nek a adatoktól és a
        rendszerlogikától való
        elválasztására az MVC (Model-View-Controller) pattern szellemében
        Az MVC (Model-View-Controller) pattern-ről lásd: http://ootips.org/mvc
        -pattern.html
        Megjegyzés: Az elnevezési konvenciók ebben a class-ban a PEP 8 elnevezési
        konvencióit követik,
        mivel ez a class többségében kívül esik a wxPython kontextusán
    """
    def __init__(self):
        self.list = []

    def get_list(self):
        """ A belső listára egy hivatkozást tesz lehetővé.
        """
        return self.list

    def add_element(self, position=0, text=""):
        """ Hozzáad egy elemet a listához, ha lehetséges
        """
        if len(self.list) >= position and position >= 0:
            self.list.insert(position, text)
        else:
            self.list.append(text)

    def del_element(self, position):
        """ Kivesz egy elemet a listából, ha lehetséges
        """
        if len(self.list) >= position and position >= 0:
            del self.list[position]

class MainWindow(wx.Frame):
    """ Az alkalmazás főablaka. Minden esetre csak egy lesz belőle.
    """
    def __init__(self, title):
        # A frame-nek 640*480-as kezdő mérete van és nem fog tudni e fölé menni
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(640, 480))
        self.SetMinSize(self.GetSize())

        # A frame különböző widget-jeinek létrehozása
        self.panel = wx.Panel(self)
        self.lblEntry = wx.StaticText(self.panel, wx.ID_ANY, "Adatbevitel")

        # Az adatbevitel felügyelete természetesen a fókusszal kezdődik
        self.txtEntry = wx.TextCtrl(self.panel, wx.ID_ANY)
        self.txtEntry.SetFocus()

        # A lista inaktiválva van, mert kezdetben üres
        self.lstListe = wx.ListBox(self.panel, wx.ID_ANY)
        self.lstListe.Enable(False)

        # A Ment gomb a lista elé jön a tabulációval történő fókuszálás listájában
        # ráadásul a program főműveletét reprezentálja, amit az <Enter>
        # billentyű lenyomása aktivál
        self.cmdEnregistrer = wx.Button(self.panel, wx.ID_ANY, "Ment")
```

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

        # A lista vezérlőgombjainak kezdetben inaktívaknak kell lenni
        self.cmdModifier = wx.Button(self.panel, wx.ID_ANY, "Módosít")
        self.cmdModifier.Enable(False)
        self.cmdSupprimer = wx.Button(self.panel, wx.ID_ANY, "Töröl")
        self.cmdSupprimer.Enable(False)
        self.cmdQuitter = wx.Button(self.panel, wx.ID_ANY, "Kilép")

        # A mnemonikus billentyűk asszociálása
        table = [(wx.ACCEL_CTRL, ord('E'), self.cmdEnregistrer.GetId()),
                (wx.ACCEL_CTRL, 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))

        # Egy karakterlánc, ami figyelmeztet a törlésre
        self.confirmSupprMess = "Valóban törölni akarja a " \
                            + " kiválasztott elemet ?"

        # Egy interface-től független lista létrehozása
        self.listObject = ListControler()

        # Az alapkiíratást és a szükséges esemény-metódus-widget összekapcsolást
        # lehetővé tevő két privát metódus
        self.__do_layout()
        self.__do_binding()
        self.modeModification = False

    def __do_layout(self):
        """ A widget-ek létrehozása és elrendezése az ablakban. Ez egy komplett
            alkalmazása a sizer-ekre vonatkozó elméletnek.
        """
        # Ez az ablak két részre van felosztva. Az egyik rész az adatbevitel TextCtrl
          widget-je számára van
        # a másik rész a többi widget számára van
        mainSizer = wx.FlexGridSizer(rows=2)
        mainSizer.AddGrowableCol(0)
        mainSizer.AddGrowableRow(1)

        # Ez a sizer kezeli a felhasználó adatbeviteli widget-jeit
        #(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)

        # Ez a sizer kezeli azt, ami a listához kapcsolódik,
        # valamint a Kilép gombot
        listSizer = wx.FlexGridSizer(cols=2, hgap=30)
        listSizer.AddGrowableCol(0)
        listSizer.AddGrowableRow(0)
        listSizer.Add(self.lstListe, 0, wx.EXPAND)

        # Ez a sizer teszi lehetővé, hogy megőrizzük a gombok oszlopokba
        # rendezettségét
        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):
        """ Itt hozzuk létre az erre az ablakra vonatkozó eseménykezelőket
        """
        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):
        """ Kattintás a "Ment" gombra
        """
        # Ha "Módosít" mód aktív, akkor módosítani kell
        # a listából kiválasztott elemet,
        if self.modeModification:
           self.listObject.del_element(self.selPos)
           self.listObject.add_element(self.selPos,
                                       self.txtEntry.GetLineText(0))
           self.modeModification = False

        # ha nem, akkor a szöveget a lista végéhez kell fűzni
        else:
           self.listObject.add_element(text=self.txtEntry.GetLineText(0))

        # A lista szinkronizálása a kontroller listájával
        self.lstListe.Set(self.listObject.get_list())

        # Az adatbeviteli mező szövegének törlése, a fókusz visszakapása
        # és a lista aktívvá válik, hogy elemet válasszunk belőle
        self.txtEntry.SetValue("")
        self.txtEntry.SetFocus()
        self.lstListe.Enable(True)

    def OnClickModifier(self, event=None):
        """ Kattintás a "Modifier" gombra
        """
        # Aktiválja a Módosítás módot és kinyeri a kiválasztott elem indexét
        self.modeModification = True
        self.selPos = self.lstListe.GetSelection()

        # Az adatbeviteli mező felveszi a kiválasztott értéket és megkapja a fókuszt
        self.txtEntry.SetValue(self.lstListe.GetStringSelection())
        self.txtEntry.SetFocus()

        # Inaktiválja a listát és a listát vezérlő widget-eket,
        # hogy elkerüljük a felhasználó okozta hibákat
        self.lstListe.Enable(False)
        self.cmdModifier.Enable(False)
        self.cmdSupprimer.Enable(False)

    def OnClickSupprimer(self, event=None):
        """ Kattintás a "Töröl" gombra
        """
        # Egy jóváhagyást kérő üzenet kiíratása
```

```
        confirmSuppr = wx.MessageDialog(self, self.confirmSupprMess,
                                        style=wx.YES_NO | wx.NO_DEFAULT)

        # A felhasználó jóváhagyására törli az elemet a listából
        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):
        """ Kattintás a "Kilép" gombra
        """
        self.Close()

    def OnListSelect(self, event=None):
        """ Egy elem kiválasztása a listából
        """
        self.cmdModifier.Enable(True)
        self.cmdSupprimer.Enable(True)

class MainApp(wx.App):
    """ Az alkalmazás főablakát hozza létre
    """
    def OnInit(self):
        frame = MainWindow(title="Listaszerkesztő")
        self.SetTopWindow(frame)
        frame.Show()
        return 1

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

11.10. Rádiógombos és jelölőnégyzetes példa (radio_check.py)

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import wx
class MainWindow(wx.Frame):
    """ Az alkalmazás főablaka. Minden esetre csak egy lesz belőle. """

    def __init__(self, title):
        # A frame-nek 640*480-as kezdő mérete van és nem fog tudni e fölé menni
        wx.Frame.__init__(self, None, wx.ID_ANY, title, size=(320, 200))
        self.SetMinSize(self.GetSize())
        self.SetMaxSize(self.GetSize())

        # A frame különböző widget-jeinek létrehozása
        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")

        # Az alapkiíratást és a szükséges esemény-metódus-widget összekapcsolást
        # lehetővé tevő két privát metódus
        self.__do_layout()
        self.__do_binding()

    def __do_layout(self):
        """ A widget-ek létrehozása és elrendezése az ablakban. Ez egy komplett
            alkalmazása a sizer-ekre vonatkozó elméletnek.
        """
        # Ez az ablak két sorra van felosztva. Az egyik a rádiógomboké
        # és a jelölőnégyzeteké, a másik az eredményé
        mainSizer = wx.FlexGridSizer(rows=2)

        # Ez a sizer kezeli a felhasználó adatbeviteli widget-jeit
        #(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):
        """ Itt hozzuk létre az erre az ablakra vonatkozó eseménykezelőket
        """
        self.Bind(wx.EVT_RADIOBOX, self.ChangeResult, self.radioBox)
        self.Bind(wx.EVT_CHECKLISTBOX, self.ChangeResult, self.checkListBox)

    def ChangeResult(self, event):
        """ Megváltoztatja az eredményt, ami a kiválasztott elemek összege
        """
        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")
        print self.checkListBox.GetSelections()

class MainApp(wx.App):
    """ Az alkalmazás főablakát hozza létre """
    def OnInit(self):
        frame = MainWindow(title="Generateur de frequence")
        self.SetTopWindow(frame)
        frame.Show()
        return 1

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


11.11. Példa képek alakalmazására (images.py)

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

class Fenetre(wx.Frame):
    """ Ablak, amiben bemutatjuk a rajzot és a képek betöltését """
    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)

        # Képek betöltése
        imgPommes = wx.Image("pomme.jpg", wx.BITMAP_TYPE_JPEG)
        imgAbricots = wx.Image("abricot.png", wx.BITMAP_TYPE_ANY)

        imgPommes = imgPommes.Mirror(horizontally=True)

        # A képek bitmap-be konvertálása a kiíratásukhoz
        bmpPommes   = wx.StaticBitmap(panel, wx.ID_ANY,
                                      wx.BitmapFromImage(imgPommes))
        bmpAbricots = wx.StaticBitmap(panel, wx.ID_ANY,
                                      wx.BitmapFromImage(imgAbricots))

        # A rajzzóna létrehozása
        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):
    """ Rajzzóna """

    def __init__(self, parent):
        wx.Panel.__init__(self, parent, wx.ID_ANY, size=(510, 80))
        # Az EVT_PAINT esemény kinyerése
        wx.EVT_PAINT(self, self.OnDessiner)

    def OnDessiner(self, event=None):
        dc = wx.PaintDC(self)
        dc.Clear()
        # Az ecset színének és méretének definiálása
        dc.SetPen(wx.Pen("RED", 3))
        # A nyíl szárának kirajzolása
        dc.DrawLine(60, 50, 60, 7)
        # A nyíl hegyének kirajzolása
        dc.DrawPolygon([(55, 7),(65,7),(60,0)], fillStyle=wx.WINDING_RULE)
        dc.DrawText("az alma jó", 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("a barack a legjobb!", 300, 30)
```

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


11.12. Példa menük használatára (menus.py)

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

class MainApp(wx.App):
    def OnInit(self):
        # A főablak generálása
        frame = MainWindow('Menüs példa')
        self.SetTopWindow(frame)
        return True

class MainWindow(wx.Frame):
    """ Ablak a menük létrehozását bemutató példa számára"""

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

        # Egyszerű címke a menükhöz kapcsolt akciók kiíratásához
        self.texte = wx.StaticText(panel, wx.ID_ANY, "Nincs akció", pos=(30, 50))

        # A menüsáv létrehozása
        self.__createMenuBar()
        self.Show(True)

    def __createMenuBar (self):
        """ privát metódus, ami a menüsávot fogja létrehozni """
        menubar = wx.MenuBar()
        menuFile = wx.Menu()
        menuEdition = wx.Menu()

        # A file menü elemeinek létrehozása és hozzáadása a menühöz
        mnuItemOpen = menuFile.Append(wx.ID_ANY, "&Megnyitás\tCtrl-M")
        menuFile.AppendSeparator()
        menuQuit = menuFile.Append(wx.ID_ANY, "&Kilépés\tCtrl-K")

        # Mac OS X alatt, a menüsáv Quit eleme menu nem képezi részét
        # a File menünek
        if "__WXMAC__" in wx.PlatformInfo:
            wx.App.SetMacExitMenuItemId(menuQuit.GetId())

        # Az Edition menu elemeinek létrehozása
        mnuItemCancel = wx.MenuItem(menuEdition, wx.ID_ANY, "Töröl\tCtrl-T")
        mnuItemRestore = wx.MenuItem(menuEdition,wx.ID_ANY,
                                     "Visszaállít\tCtrl+Shift-V")

        # Egy almenü létrehozása
        sousMenuAction = wx.Menu();
        mnuItemAction1 = sousMenuAction.Append(wx.ID_ANY, "Akció 1")
        mnuItemAction2 = sousMenuAction.Append(wx.ID_ANY, "Akció 2")
        mnuItemAction3 = sousMenuAction.Append(wx.ID_ANY, "Akció 3")

        # A menüpontok hozzáadása az Edition menühöz
        menuEdition.AppendItem(mnuItemCancel)
        menuEdition.AppendItem(mnuItemRestore)
        menuEdition.AppendSeparator()
        menuEdition.AppendMenu(wx.ID_ANY, "Action", sousMenuAction) # Almenü
                                                                    # hozzáadása
```

```
        # A menük hozzáadása a menüsávhoz
        menubar.Append(menuFile, "File")
        menubar.Append(menuEdition, "Edit")
        self.SetMenuBar(menubar)

        # Események kapcsolása
        self.Bind(wx.EVT_MENU, self.OnOpen, mnuItemOpen)
        self.Bind(wx.EVT_MENU, self.OnQuit, menuQuit)
        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, mnuItemCancel)
        self.Bind(wx.EVT_MENU, self.OnNonImplemente, mnuItemRestore)

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

    def OnOpen (self, evt):
        self.texte.SetLabel( "Az OnOpen akció hívása" )

    def OnAction (self, evt):
        itemAction = self.GetMenuBar().FindItemById( evt.GetId())
        label = itemAction.GetText()
        self.texte.SetLabel (u"Az %s hívása" % label)

    def OnNonImplemente (self, evt):
        self.texte.SetLabel ("Nincs implementálva akció")

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


12. Függelékek

12.1. A. függelék : A wxPython és a PyQt összehasonlítása

A következő részben megkíséreljük a wxPython és a PyQt könyvtárakat röviden és objektíven összehasonlítani. Ez a két - különböző elveken alapuló - könyvtár lehetővé teszi grafikus interface-ek egyszerű fejlesztését.

Az első különbség, amit észreveszünk a Qt és a wxWidgets (C++ alapú könyvtárak) között, a dokumentációjukra vonatkozik. A Qt profibbnak tűnik, mert egy cég áll mögötte, míg a wxWidgets egy közösségi fejlesztésű könyvtár. Következésként a Qt dokumentációval jóval ellátottabb és a dokumentációja jóval komplettebb, mint a wxWidgets-é.

Mindazonáltal, pillanatnyilag sem a PyQt-nak, sem a wxPythonnak nincs igazi referencia dokumentációja, mert rendkívül egyszerű az áttérés a C++-ról a Pythonra. Minden esetre meg kell, hogy jegyezzük, a wxWidgets dokumentációja tartalmaz bizonyos információkat a wxPythonra vonatkozóan, amikor például a wxPython metódusai egy kicsit különböznek a C++ metódusaitól. Ráadásul van egy teljesen a wxPythonnak szentelt könyv.

A Qt-nek egy rendkívül hatékony és praktikus eseménykezelő rendszere van. A signal- és slot-rendszer lehetővé teszi egy eseményhez (signal) több akció (slot) hozzárendelését, és egy akciót több esemény aktiválhat. Sajnos a wxPython nem engedi meg, hogy ugyanaz az esemény több metódust hívjon.

A wxWidgets-tel – ami csak egy interface-könyvtár – szemben a Qt-nek van számos egyéb class-a, amik nem az interface-ek kezelésével függenek össze. Egyebek között megemlítjük az adatbázis-, az XML-kezelést, hálózati kapcsolatok létrehozásának lehetőségét, stb. A Qt tehát jóval több, mint egy egyszerű interface-könyvtár. Ennek ellenére megőrzi az alkalmazás egyszerűségét.

A wxWidgets egy nagyon hatékony rendszert ad a programozó kezébe a grafikus elemek elhelyezését illetően. A sizer-ek, amiket kezdetben kicsit nehéz megérteni, lehetővé teszik az elemek elhelyezésének és átméretezésének egyszerű kezelését, például azt definiálva, hogy az elemeknek meg kell őrizniük méretarányaikat, hogy bizonyos elemek méretei bizonyos határig nőhetnek vagy zsugorodhatnak, stb. Ilyen lehetőségek a Qt-ben is vannak, a Qt Designer alkalmazásával rendkívül egyszerűvé válik egy alkalmazás interface-ének az elkészítése és tesztelése.

A wxPython telepítése jóval egyszerűbbnek tűnik, mint a Qt-é. Ennek bizonyosan abból a tényből kell következni, hogy a QT könyvtár jóval alacsonyabb szintű, mint a wxWidgets. A Qt-nek saját kiíratók rutinjai vannak, míg a wxWidgets annak az operációs rendszernek a kiíratók rutinjaira támaszkodik, amely alatt fut.

E rövid összehasonlításból azt a következtetést vonhatjuk le, hogy nem tűnik úgy, hogy az egyik könyvtár jobb lenne, mint a másik. Minden attól függ, hogy mit akarunk csinálni. Egy olyan alkalmazáshoz, amihez csak egy grafikus interface kell, ahhoz kiválóan megy a wxPython könyvtár, viszont ha egy összetettebb alkalmazást akarunk készíteni, például hálózatkezeléssel, akkor a QT egy jobb választás lesz.

Jegyezzük meg viszont, hogy wxPythonnal tökéletesen lehetséges a hálózattal kommunikáló alkalmazások fejlesztése, a Pythonnak vannak a hálózatkezelést lehetővé tevő class-ai. A Qt egyetlen előnye ezen a ponton az, hogy csak egyetlen könyvtárra van szüksége. Az ezzel járó hátrányokat. mint a program betöltésének nehézkessége, a Qt még a 4. verzióban is a könyvtár modulokra való felosztásával oldja meg.

12.2. B. függelék : Python script OpenOffice 2 -höz

Kiegészítésként megadjuk a "macro OO2.org" szkript forráskódját, ami bold-ban íratja ki az ebben a dokumentumban lévő Python kódok kulcsszavait.

```
# -*- 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,
```

Vigyázat : A szkript végrehajtása olyan nagyméretű dokumentumokon, mint amilyen ez, néha az OpenOffice.org 2 kiakadásához vezet.

