



بناء التطبيقات مع بايثون بالاعتماد على SQLite و GTK

```
def main():
75     window = gtk.GtkWindow(gtk.WINDOW_TOPLEVEL)
76     window.set_name ("Test Input")
77
78     vbox = gtk.GtkVBox(gtk.FALSE, 0)
79     window.add(vbox)
80     vbox.show()
81
82     window.connect("destroy", gtk.mainquit)
83
84     # Create the drawing area
85     drawing_area = gtk.GtkDrawingArea()
86     drawing_area.size(200, 200)
87     vbox.pack_start(drawing_area, gtk.TRUE, gtk.TRUE, 0)
88
89     drawing_area.show()
90
91     # Signals used to handle backing pixmap
92     drawing_area.connect("expose_event", expose_event)
93     drawing_area.connect("configure_event", configure_event)
94
95     # Event signals
96     drawing_area.connect("motion_notify_event", motion_notify_event)
97     drawing_area.connect("button_press_event", button_press_event)
```

المحتويات

4	<u>مقدمة الطبعة الثانية</u>
7	<u>مقدمة الطبعة الأولى</u>
11	<u>شكرو تقدير</u>
12	<u>إهداء</u>
13	<u>الفصل الاول : مدخل إلى GTK</u>
14	<u>مقدمه</u>
14	<u>ما هي GTK؟</u>
15	<u>نظرة سريعة على ماضي GTK</u>
16	<u>قبل البدء</u>
17	<u>المثال الاول : مرحبا بالعالم</u>
19	<u>الادوات Widget</u>
19	<u>ما هي الادوات Widget</u>
19	<u>الادوات مع ال PyGTK</u>
19	<u>البداهه</u>
26	<u>الصناديق</u>
26	<u>مقدمه</u>
27	<u>Vbox و HBox</u>
28	<u>استخدام الصناديق عمليا</u>

33	الإشارات و الدوال <u>Callback</u>	ص 33
33	مقدمه.....	ص 33
34	الاشارات مع <u>PyGTK</u>	ص 34
41	المثال الثاني : برنامج تحويل درجة الحرارة.....	ص 41
57	المثال الثالث : آله حاسبه.....	ص 57
72	مدخل إلى <u>Glade</u>	ص 72
74	التصميم مع <u>Glade</u>	ص 74
79	استخدام ما تم تصميمه مع <u>Glade</u>	ص 79
84	استخدام الاشارات مع ما تم تصميمه مع <u>Glade</u>	ص 84
93	نقل برنامج تحويل درجة الحرارة إلى <u>Glade</u>	ص 93
103	نقل الآله الحاسبه إلى <u>Glade</u>	ص 103
115	الفصل الثاني : مدخل إلى <u>SQLite</u>	ص 115
116	مقدمه.....	ص 116
117	الدوال الاساسيه.....	ص 117
120	المثال الاول : برنامج لتخزين الاسماء و عرضها.....	ص 120
144	الفصل الثالث : إنشاء مشروع.....	ص 144
145	برنامج إدارة بيانات موظفين في شركه.....	ص 145
179	ملحق أ : وصلات.....	ص 179
181	قائمة المصادر.....	ص 181

مقدمة الطبعة الثانية

بسم الله الرحمن الرحيم، و الصلاة و السلام على اشرف المرسلين سيدنا و مولانا ابا القاسم محمد و على آله الطيبين الطاهرين و رضوان الله على الصحابة المنتجبين اما بعد.

{ سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنْتَ الْعَلِيمُ الْحَكِيمُ }

طُرِحَتْ الطبعة الأولى من هذا الكتاب في عام 2009، مرّت الأعوام و في فتره من الفترات و لبعض الظروف أغلقت مدونتي التي كانت تحتوي على رابط تحميل مباشر لهذا الكتاب، بعد فتره أعدت إفتتاح المدونة و أعدت رفع الكتاب لمن أراد الإستفادة منه، لاحظت من خلال إحصائيات زوار الموقع إنّه على الرغم من مرور ما يقارب الثلاث سنوات على طرح الكتاب إلا إنه لا يزال مطلوباً و هنا قررت العمل على الطبعة الثانية التي بين يديك و هي تحسين على الطبعة الأولى على أمل أن تكون أكثر فائدة ووضوحاً.

من أهم التحسينات على الطبعة الثانية هي الإنتقال من إستخدام مكتبة Libglade (التي شرحت في الطبعة الأولى) إلى إستخدام مكتبة GtkBuilder و التي بدأ المبرمجون بالإنتقال إليها، بالإضافة إلى ذلك تم تحديث لقطات الشاشة و إضافة لقطات شاشة جديدة توضّح شكل البرامج التي نبنها في أمثلة الكتاب، كما قمت بتحسين عام للشيفرات بقدر الإمكان و إختبارها على الإصدار 2.7 من بايثون، هذا بالإضافة إلى بعض التوضيحات الأخرى في نص الكتاب في الأماكن التي وجدت إنها غامضة بعض الشيء و تحتاج للمزيد من التوضيح.

لمن يقرأ الكتاب لأول مرّه فإنّه لا بد من التنبيه إلى إنّه يتطلب معرفة سابقة

في لغة بايثون و إلمام في لغة SQL للتعامل مع قواعد البيانات.

الكتاب حُر و مجاني يقع تحت رخصة GNU FDL و جميع شيفرات الكتاب تحت رخصة GNU LGPL الحُرّه، و كالعادة تم إستخدام البرامج الحُرّه من أجل إنجازّه - محرر LeafPad لتحريره، برنامج WikkaWiki لتخزينه و التعامل معه اثناء كتابته، برنامج LibreOffice Writer من اجل تنسيقه و وضعه في صورته النهائيه من اجل نشره. و كلها تعمل تحت مظلة نظام التشغيل GNU/Linux توزيعه OpenSuSE، شكرا لجهود مجتمع البرمجيات الحُرّه :-).

يُمكنك الحصول على شيفرات جميع الأمثلة من خلال مستودع Git على العنوان التالي : <http://github.com/MaaSTaaR/PyGTK-Book-Examples>

لأي تصحيح أو سؤال أو إقتراح يخص الكتاب يُمكنك التوا صل معي عن طريق مراسلتي من خلال مدونتي الشخصية على العنوان التالي <http://www.maastaar.com> أو عن طريق حساب تويتر @MaaSTaaR

هذا و الله ولي التوفيق.

مقدمة الطبعة الأولى

بسم الله الرحمن الرحيم، و الصلاة و السلام على اشرف المرسلين سيدنا و مولانا ابا القاسم محمد و على آله الطيبين الطاهرين و رضوان الله على الصحابة المنتجبين اما بعد.

منذ فتره و انا افكر بكتاب ينتفع به من يريد التعلم، و بدأت اقلب بالمواضيع حتى و صلت إلى لغة البايثون، دقت النظر في المجتمع العربي على الانترنت فوجدت أنّ هناك كتابان باللغة العربية يتحدثان عن هذه اللغة للمبتدئين، فقررت أن لا اكتب عن نفس الموضوع الذي سبقوني به، لانه إن حصل ذلك لن يحقق الكتاب هدفه، و في الحقيقة مللنا من كثرة الكتب التي تتحدث عن الاساسيات فقط خصوصا بالبرمجه، فما اكثر الكتب التي تشرح اساسيات العديد من لغات البرمجه متغاضية عن المواضيع المتقدمه فيها، لذا و بعد كل هذا قررت التحدث عن لغة البايثون و لكن في مواضيع متقدمه فيها، فأخترت الكتابه عن GTK و SQLite مع البايثون، حتى يتمكن من ينتهي من قراءه هذا الكتاب بناء تطبيقات متكامله ذات واجهات رسوميه بالاعتماد على قواعد البيانات من اجل ادارة البيانات.

بالنسبه للمصطلحات، فحاولت ترجمتها بقدر الامكان مع الابقاء على المعنى الصحيح و ان كان هذا المعنى ينافي معنى الكلمه حرفيا، و ارجو أن لا يلومني البعض على ترجمتي لهذه المصطلحات كما حدث لي من قبل (-:). كما أنّي وضحت مع كل مصطلح قمت بترجمته الكلمه الاصليه باللغه الانجليزيه، حتى اسهل على من اراد التبخر و البحث اكثر و اقرب الصوره لمن لم يتقبل ترجمتي للمصطلح.

هناك نقطه هامه جدا لا بد من ذكرها، و هي أنّ هذا الكتاب يتطلب معرفة

سابقه في لغة بايثون، لن يعلمك الكتاب ما هي البرمجة او ما هي بايثون و كيفية التعامل معها، من الافضل مراجعة الكتب المتخصصة للمبتدئين في هذا المجال، كذلك يفترض هذا الكتاب أنّ لديك إلمام في لغة SQL من اجل التعامل مع قواعد البيانات.

هذا الكتاب مجاني حُر يقع تحت رخصة GNU FDL، و تم استخدام البرامج الحرّة من اجل انجازه - محرر LeafPad لتحريره، برنامج WikkaWiki لتخزينه و التعامل معه اثناء كتابته، برنامج OpenOffice Writer من اجل تنسيقه و وضعه في صورته النهائيه من اجل نشره، و كلها تعمل تحت مظلة نظام التشغيل GNU/Linux توزيعه OpenSuSE، اسمحولي برفع القبعه لمجتمع المصادر الحرّه :-).

لأي سؤال حول الكتاب، او حول احد مواضيع الكتاب و شيفراته، او تصحيح مهما كان نوعه يمكنك التوجه إلى مدونتي على العنوان التالي <http://www.maastaar.com> ثم اذهب إلى صفحة "راسلني" و اكتب استفسارك او اقتراحك او تصحيحك و سوف احاول الرد بأسرع وقت ممكن ان شاء الله، يسعدني أن تراسلوني بأسئلة حول مواضيع الكتاب لأنها حتما ستزيد من خبرتي و ستساعدني في الطبعه الثانيه من هذا الكتاب الذي افكر بوضع ملحق لأسئلة القراء و اجوبتها ان شاء الله :-).

جميع الشيفرات البرمجيّه في هذا الكتاب مُجربه على الاصدار 2.6 من بايثون، و جميعها تخضع للرخصه الحرّه GNU LGPL.

للحصول على الخطوط المُستخدمه في الكتاب راجع العنوان التالي :
<http://wiki.arabeyes.org/خطوط>

اما الخط المُستخدم في الشيفرات البرمجيه فراجع العنوان التالي :
[/http://www.proggyfonts.com](http://www.proggyfonts.com)

هذا و نرجوا من الله سبحانه و تعالى ان يتقبل منّا هذا العمل البسيط و
الحمد لله رب العالمين .

شكر و تقدير

أشكر مجتمع البرمجيات الحرة على جهودهم الجبارة التي يبذلونها في إنتاج مثل هذه الأدوات الرائعة.

إهداء

إلى والديّ.

الفصل الأول

مدخل إلى GTK

مقدمة

ما هي GTK

GTK عبارة عن مكتبة برمجية لتصميم واجهات مستخدم رسومية GUI، فبدلاً من بناء برامجنا على شكل سطر اوامر يمكننا بناءه على الشكل الحديث للبرامج عن طريق واجهة مستخدم رسومية، بمعنى انه يمكن بناء واجهات البرامج ووضع الازرار و مربعات النص و غيرها من الامور مع ما يناسب برنامجنا، كلمة GTK اختصار للجمله GIMP Toolkit، تقع هذه المكتبة تحت رخصة GNU LGPL و تتميز بكفاءتها مع اللغة العربية، تتوفر مكتبة GTK مع مجموعه كبيره من لغات البرمجه مثل سي و سي++ و جافا و PHP و بالطبع بايثون و التي سوف نتعامل مع الـ GTK من خلالها :-)، تسمى المكتبة التي تدعم GTK في البايثون بـ PyGTK، الجدير بالذكر ان مكتبة GTK هي المكتبة الاساسيه التي يستخدمها سطح مكتب GNOME لبناء تطبيقاته وواجهته، بينما يستخدم سطح المكتب KDE مكتبة Qt لبناء الواجهات الرسومية، و من ابرز البرامج التي تم تصميمها باستخدام الـ GTK هو برنامج GIMP برنامج الرسم المشهور، و AbiWord معالج النصوص و مجموعه لا يستهان بها من البرامج (راجع الملحق أ للحصول على و صلات لمواقع بهذا الخصوص)، ولا بد من الملاحظه ان GTK تعمل على نظام التشغيل غنول لينكس و آبل ماكنتوش و مايكروسوفت وندوز، هذا يعني امكانية كتابة برنامج واحد ليعمل على الانظمه الثلاث.

نظرة سريعة على ماضي GTK

بدأت GTK اساسا في عام 1996 من اجل برنامج الرسم GIMP على يد Spencer Kimball و Peter Mattis و Josh MacDonald عندما قرروا الاستغناء عن Motif (وهي مكتبة اخرى لبناء واجهات مستخدم رسومية) و هذا هو سبب تسمية GTK بـ GIMP Toolkit.

تعتبر مكتبة GTK احد اجزاء مشروع GNU و تُستخدم الآن بشكل اساسي في نظم يونكس (مثل غنول لينكس و FreeBSD و غيرها، و كما ذكرنا مسبقا انه يعمل على نظم التشغيل الاخرى مثل مايكرو سوفت وندوز).

قبل البدء

أولاً : قبل ان نبدأ بأي شئ لا بد من تثبيت مكتبة PyGTK على حاسوبنا، يمكنك الحصول على هذه المكتبة من موقعها الرسمي <http://www.pygtk.org>.

ثانياً : في كل ملف Python نود استخدام PyGTK فيه، لا بد من كتابة الاوامر التاليه :

```
import pygtk
pygtk.require('2.0')
import gtk
```

و كما ذكرت في مقدمة الكتاب، إن من يقرأ هذا الكتاب لا بد أن يكون مُلمّاً بلغة Python، وهذا يعني إن الشيفره المذكوره في الاعلى مفهومه و ما هي إلا استدعاء لمكتبتين، أما السطر الثاني فهو يلزم مكتبة pygtk على اختيار الاصدار 2.0 فما فوق.

المثال الاول : مرحباً بالعالم

حتى نبدأ بالدراسة العملية للمفاهيم الأساسية والتي سوف تساعدنا على بناء التطبيقات يجب أن نبدأ بمثال بسيط يستخدم هذه المفاهيم الأساسية ثم نلج في شرح المفاهيم إستناداً إلى هذا المثال. كالعاده سوف نبدأ ببرنامج "مرحباً بالعالم!" او كما نعرفه باسم Hello World.!

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

# ... #

window = gtk.Window( gtk.WINDOW_TOPLEVEL
);

window.set_title( "مرحباً بالعالم" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );
```

```
# ... #  
hello_world_button = gtk.Button( "مرحباً بالعالم"  
);  
hello_world_button.show();  
  
# ... #  
window.add( hello_world_button );  
window.show();  
gtk.main();
```

لا تدقق في الشيفره كثيراً :-)، إنتقل للجزء التالي من الكتاب و سوف يتم شرحها.

الادوات Widget

ما هي الادوات Widget

الادوات هي افضل ترجمه وجدتها للمصطلح Widget، تُعتبر الادوات الجزء الاساسي من اي مكتبه لبناء واجهات المستخدم الرسومي، و بالطبع من ضمن هذه المكتبات مكتبة GTK، الادوات هي كل ما يُعرض بواجهة البرنامج، مثلا نافذة البرنامج الرئيسية تعتبر من الادوات، كذلك مربع النص الذي نقوم بتعبئة البيانات في داخله يعتبر من الادوات، و كل شئ موجود داخل نافذة البرنامج يعتبر من الادوات مربع النص، الازرار، القوائم المنسدله، قائمة اختيار الملفات و غيرها، في مثالنا السابق كان لدينا اداتين، النافذه الرئيسييه، و الزر المكتوب عليه "مرحبا بالعالم".

الادوات مع ال PyGTK

البدايه

في PyGTK لكل اداة من الادوات التي يقدمها فئة (Class) خاصه بها و من خلال إنشاء كائن (Object) من هذه الفئة فإننا نقوم بإنشاء الاداة و من خلال الكائن نتحكم في الاداة التي قمنا بإنشاءها، حسب مثالنا السابق قمنا بإنشاء نافذه و وضعنا داخل هذه النافذه زر، لنأخذ اولا السطر الذي قمنا بإنشاء النافذه من خلاله و هو التالي :

```
window = gtk.Window( gtk.WINDOW_TOPLEVEL
);
```

حسناً كما تلاحظ في الشيفره السابقه، قمنا بوضع متغير اسمه window، و استخدمنا الفئة gtk.Window من اجل إنشاء النافذه الرئيسييه و كلفنا المتغير window بها و بالتالي أصبح كائنا (Object) يمكننا من خلاله

التحكم بالنافذة (لا بد من الملاحظه ان تكون قد درست ال OOP مع ال Python حتى تتمكن من استيعاب ما يُشرح). و كما ذكرنا مسبقا إن كل اداة في GTK يكون لها فئة خاصة بها، هذا يعني أنّ لدينا العديد من الفئات غير الفئة `gtk.Window`، مثلا لدينا الفئة `gtk.Button` للازره و `gtk.Entry` لمربعات النصوص و عشرات الفئات التي تخدم مجموعه كبيره من المجالات، بالطبع لن نتمكن من ذكر جميع هذه الفئات في هذا الكتاب و لكننا سنذكر الاساسيه منها حتى نتعلم طريقة الكتابه، و بعدها يمكنك التبهر عن طريق قراءة الدليل الرسمي الخاص بـ PyGTK و الذي يشرح جميع الفئات المتوفره.

على كل حال نعود لمثالنا، كما أسلفنا إنّ المتغير `window` أصبح كائناً (Object)، يمكننا من خلال هذا الكائن التحكم بالنافذة التي أنشأناها من خلال مجموعه من الدوال التي توفرها لنا الفئة `gtk.Window`. بالطبع لكل اداة مجموعه من الدوال التي تخصها و التي تعطيك التحكم الكامل في الاداة التي تقوم بإنشاءها، ننتقل الآن إلى السطر التالي في مثالنا و هو :

```
window.set_title( "مرحباً بالعالم" );
```

الداله `set_title` هي إحدى الدوال التي تُقدمها الفئة `gtk.Window` و التي تساعدك على التحكم في الاداة التي قمت بإنشاءها، تقوم هذه الداله بتغيير عنوان النافذة إلى محتوى البارامتر المُمرر لها و في حالتنا هذه فإن القيمة هي "مرحبا بالعالم"، بالطبع هناك العشرات من الدوال التي يقدمها الصنف `gtk.Window` يمكنك القراءة عنها في دليل PyGTK الرسمي، لنأخذ مثلا سريعا على احد هذه الدوال، الداله `resize` تقوم بتغيير حجم النافذه، قم بإضافة السطر التالي اسفل السطر المدروس :

```
window.resize( 200, 200 );
```

لاحظ أنّ حجم النافذة تغير عن المره السابقه و إنّهُ أصبح اكبر، البارامتر الاول لهذه الداله هو عرض النافذه، و البارامتر الثاني هو الارتفاع.

الآن تجاهل السطرين التاليين لأننا سندرسهما في موضع آخر و إذهب إلى السطر الذي يليهما، وهو

```
hello_world_button = gtk.Button( "مرحباً بالعالم"  
);
```

وفقاً لما اسلفنا من شرح لابد و انك عرفت وظيفة هذا السطر، يقوم هذا السطر بإنشاء زر جديد و يجعل المتغير `hello_world_button` كائن لهذا الزر و كما تلاحظ أنّ البارامتر هو النص الذي سيُطبع على الزر، و كما هو الحال مع النافذه التي انشأناها هناك مجموعه من الدوال التي تقدمها الفئة `gtk.Button` و التي تعطينا التحكم في الزر الذي أنشأناه، لنأخذ مثلاً الداله `set_label` و التي تقوم بتغيير النص الموجود على الزر، مثال :

```
hello_world_button.set_label( "تم تغيير النص"  
);
```

يقوم هذا السطر بتغيير النص الموجود على الزر من "مرحباً بالعالم" إلى "تم تغيير النص"

نتنقل إلى السطر التالي وهو :

```
hello_world_button.show();
```

بعدهما قمنا " بإنشاء " الزر، و قمنا بالتعديل عليه بما يناسب برنامجنا (مثل اختيار النص الذي يظهر على الزر و غيره من هذه الامور) لا بد من "عرض" الزر و هذا يتم باستخدام الداله show، نستخدم الداله show مع جميع الادوات تقريبا حتى يتم عرض الاداة بعد إنشاءها.

نتقل الآن إلى السطر التالي :

```
window.add( hello_world_button );
```

بعدهما اظهرنا الزر في السطر السابق لا بد من اضافته إلى النافذه الرئيسيه، و هذا ما نقوم به في السطر الذي نقوم بشرحه الآن حيث نستخدم الداله add لهذا الغرض، في الحقيقه لا يمكننا استخدام هذه الداله إلا مره واحده فقط في برنامجنا، اي اننا لا نستطيع إلا اضافه اداة واحد فقط إلى النافذه الرئيسيه، قد تتساءل في هذه الحاله كيف يمكننا وضع اكثر من اداة في نافذتنا الرئيسيه؟

لنفرض إننا نريد تصميم برنامج لتحويل القياسات، في هذه الحاله سوف نحتاج إلى مجموعه من العناصر، مثلا مربع نصوص لإستقبال الرقم المطلوب تحويله، زر ليضغط عليه المستخدم بعد كتابة المعلومات، سطر نصي لطباعة النتائج، في هذه الحاله كيف يمكننا اضافه جميع هذه الادوات على نافذتي الرئيسيه؟ و في الحقيقه لا يمكننا استدعاء داله الاضافه add إلا مره واحده فقط!، الجواب على هذا السؤال هو باستخدام الاداتين VBox و HBox او ما سوف نطلق عليهما اسم الصناديق، سوف نشرح كيفية استخدام الصناديق فيما بعد.

السطر التالي :

```
window.show();
```

كما ذكرنا سابقاً، الدالة `show` تقوم بعرض الاداة بعد إنشائها و هذا فعلناه مع الزر الذي قمنا بإضافته، أنشأناه اولاً ثم عرضناه، و هذا الذي يجب ان يحصل مع اغلب الادوات، و كما نعلم ان النافذه الرئيسيّه تعتبر من ضمن الادوات و بالتالي لابد من عرضها بعد إنشائها، و هذا ما يتم في السطر الذي ندرسه الآن.

نتنقل إلى السطر الاخير وهو :

```
gtk.main();
```

و هذا السطر لابد من كتابته في جميع برامجنا في النهايه، لأنه هو المسؤول عن عرض ما قمنا بتصميمه.

الصناديق

مقدمه

في اثناء شرحنا السابق لشيفرة "مرحبا بالعالم" ظهر لدينا إشكال، و هو عدم إمكانية استدعاء الداله add اكثر من مره واحده، و كما نعلم هذه الداله تقوم بإضافة الادوات التي ننشؤها إلى النافذه معينه، و اذا عرفنا انه لا يمكننا استدعاءها إلا مره واحده هذا يعني عدم إمكانية إضافة اكثر من اداة واحده في برنامجنا، ذكرنا في ذاك الموضوع إنّ الحل مع الصناديق.

الصناديق HBox و VBox هي ادوات توفرها PyGTK، تعتبر الصناديق من اهم الادوات التي سوف نستخدمها في جميع برامجنا التي تحتوي على أكثر من اداة، لأوضح الفكرة أكثر : في PyGTK يجب أن تكون كل اداة في صندوق لوحده، مثلا لنفرض اننا نريد تصميم واجهه تحتوي على مربع نص و زر واحد، افتراضيا في النافذه التي ننشئها في PyGTK يمكننا إضافة اداة واحده فقط هذا يعني اننا لن نتمكن من اضافه اداة اخرى، لحل هذه المشكله لا بد من اضافه صناديق جديده و وضع الادوات فيها، هذا يعني اننا سوف ننشئ صندوقين اثنين نضع في واحد منهما مربع النص و نضع في الآخر الزر.

VBox و HBox

يوجد لدينا نوعين من الصناديق في PyGTK، النوع الاول هو HBox و هذا النوع يقوم بإنشاء الصناديق بشكل افقي، اما النوع الثاني هو VBox و الذي يقوم بإنشاء الصناديق بشكل عمودي، حسنا الآن اذا اردنا وضع الزر بجانب صندوق النص سوف نقوم باستخدام HBox، اما اذا اردنا وضع الزر في اسفل مربع النص (او العكس) سوف نستخدم VBox، الجدير بالذكر انه يمكننا وضع صناديق داخل صناديق.

لنرى الآن كيفية تعريف HBox :

```
gtk.HBox( homogeneous = False, spacing=0 )
```

اما تعريف VBox :

```
gtk.VBox( homogeneous = False, spacing = 0 )
```

البارامتر الاول `homogeneous` يعني اعطاء مساحات متساوية لما يقع داخل الصندوق اذا كانت `True`. البارامتر الثاني `spacing` و هي المسافة بين ما يقع داخل الصندوق بالبيكسل.

VBox و HBox يوفران داله و هي `pack_start`. حيث تقوم هذه الداله بوضع اداة معينه داخل صندوق معين مُعرّف مسبقا.

إستخدام الصناديق عمليا

نعود لمثالنا "مرحبا بالعالم"، اذا اردنا الآن اضافة زر آخر بجانب الزر القديم ما الذي يجب فعله؟، نعم جوابك صحيح :-) سوف نستخدم

HBox، الشيفره التاليه تحقق لنا ما أردنا :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

# ... #

window = gtk.Window( gtk.WINDOW_TOPLEVEL
);

window.set_title( "مرحباً بالعالم" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

# ... #

box1 = gtk.HBox(); # New line
window.add( box1 ); # New line

# ... #
```



```

hello_world_button = gtk.Button( "مرحباً بالعالم"
);
box1.pack_start( hello_world_button ); #
New line
hello_world_button.show();

# ... #

hello_world_button_2 = gtk.Button( "الزر
الثاني" ); # New line
box1.pack_start( hello_world_button_2 );
# New line
hello_world_button_2.show(); # New line

# ... #

box1.show(); # New line
window.show();

# ... #

gtk.main();

```

لاحظ انني وضعت التعليق "# *New line*" عند الاسطر الجديده التي قمت بإضافتها على مثالنا الاول.

لنبدأ بشرح الاسطر الجديده، اولاً :

```

box1 = gtk.HBox(); # New line
window.add( box1 ); # New line

```

في السطر الاول أضفنا الصندوق الذي سنضع فيه الاداتين، لاحظ أنّ هذا الصندوق من النوع الافقي لأننا نريد وضع الزر الجديد بجانب الزر القديم، لو كنا نريد وضع الزر الجديد اسفل الزر القديم لأستخدمنا صندوق من النوع VBox، كما إنّنا لم نمرر أي بارامترات لأننا نريد القيم الافتراضية للبارامترات و التي تحدثنا عنها عندما وضعنا تعريف HBox و VBox.

في السطر التالي قمنا بإضافة الصندوق الرئيسي إلى النافذه، لاحظ اننا حذفنا هذا السطر من اسفل الملف و اضفناه هنا.

نتقل إلى الجزء الذي يليه من الشيفره :

```
hello_world_button = gtk.Button( "مرحباً بالعالم"  
);  
box1.pack_start( hello_world_button ); #  
New line  
hello_world_button.show();
```

من خلال هذه الاسطر قمنا بإنشاء الزر و عرضه، هل تتذكر المثال السابق؟ هذه الشيفره من المثال السابق (-:)، كل ما قمنا به هو إضافة السطر الجديد و هو السطر الثاني، من خلال هذا السطر إضافة الاداة hello_world_button داخل الصندوق، سوف نستخدم الداله pack_start دائما من اجل إضافة الادوات داخل الصناديق، و على اختلاف نوع الصندوق سواء كان صندوق VBox او HBox فلا يوجد فرق فالداله نفسها.

نتقل إلى الاسطر التاليه :

```
hello_world_button_2 = gtk.Button( " الزر  
الثاني" ); # New line  
box1.pack_start( hello_world_button_2 );  
# New line  
hello_world_button_2.show(); # New line
```

لا يوجد شيء جديد هنا لاننا شرحنا معنى هذه الاسطر مسبقاً، هذه الاسطر تقوم بإضافة زر جديد بجانب الزر القديم، السطر الاول يقوم بإنشاء الزر، السطر الثاني يقوم بإضافة الزر إلى الصندوق، و السطر الاخير يقوم بإظهار الزر.

نتنقل إلى آخر سطر جديد قمنا بإضافته :

```
box1.show(); # New line
```

في هذا السطر أظهرنا الصندوق الذي وضعنا ادواتنا فيه، لاحظ انه يمكننا إظهار الاداة في اي مكان، مثلاً لا يوجد مشكلة لو قمنا بإظهار الصندوق بعد إنشائه مباشرة في مثالنا هذا، ببساطه يمكننا استدعاء الداله `show` و إظهار اداتنا في اي مكان، المهم ان ننشئ الاداة اولاً.

الإشارات و الدوال Callback

مقدمة

في بادئ الأمر درسنا مفهوم الادوات (Widget) و كما تعلم إنّه مفهوم مهم جداً، المفهوم الآخر و الذي يجب إستيعابه بشكل جيد هو "الإشارات" (Signals).

تخيل معي ان برنامجنا يحتوي على زر مكتوب عليه "اضغط هنا" عندما يضغط المستخدم على الزر لن يحدث اي شيء! لاننا وضعنا الزر ضمن النافذه لا اكثر، اذا اردنا من الزر أن يقوم بعمل مفيد سوف نستخدم شيئين هما الإشارات (Signals) و الدوال (Callback).

عندما يضغط المستخدم على الزر تُعتبر هذه الضغطه اشاره (أو حَدَث) و تؤدي إلى تنفيذ داله (Callback) تؤدي هذه الداله وظيفه معينه، مثلاً اغلاق البرنامج.

الإشارات مع PyGTK

هل تتذكر السطرين الذين تجاهلناهما عندما بدأنا بشرح مثالنا الأول "مرحبا بالعالم"؟ نعم لا بد و إنك تتذكرهما (-):

```
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );
```

هذان السطران تطبيقاً للإشارات، مع أي أداة ننشؤها يمكننا استخدام الدالة `connect` حتى نجعل الأداة تقوم بعمل مفيد، في مثالنا الحالي قمنا بربط النافذة بحدثين، الحدث الأول يسمى `delete_event` و الثاني يُسمى `destroy`، نستخدم هذين الحدثين مع النافذة الرئيسيه فقط لانهما يساعدان على إغلاق البرنامج عندما يطلب المستخدم ذلك، عندما يتم تنفيذ الحدث الأول تستدعي الدالة `delete_event` و التي عرفناها مسبقا، و عندما يتم تنفيذ الحدث الثاني تستدعي الدالة `destroy`. الدالتان `delete_event` و `destroy` هما دالتان من نوع `callback`.

```
def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();
```

هذا النوع من الدوال يستقبل على الأقل بارامترين كما تلاحظ في تعريفنا لهاتين الدالتين، حتى تتضح الصورة بشكل أكبر سنطور مثال "مرحبا بالعالم"، بحيث اذا ضغطنا على زر "مرحبا بالعالم" تظهر لنا نافذة جديده

تحتوي على نص "هذا برنامجي الاول مع PyGTK".

الشيفره التاليه ستقوم بالمطلوب :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

def the_new_window( widget, data = None ):
    new = gtk.Window();
    new.set_title( "نافذتنا الجديده" );

    msg = gtk.Label( " هذا برنامجي الاول مع PyGTK"
);
    msg.show();

    new.add( msg );

    new.show();

# ... #

window = gtk.Window();
```

```

# ... #

window.set_title( "مرحباً بالعالم" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

# ... #

box1 = gtk.HBox();
window.add( box1 );

# ... #

hello_world_button = gtk.Button( "مرحباً بالعالم"
);
hello_world_button.connect( 'clicked',
the_new_window ); # New line
box1.pack_start( hello_world_button );
hello_world_button.show();

# ... #

hello_world_button_2 = gtk.Button( "الزر
الثاني" );
box1.pack_start( hello_world_button_2 );
hello_world_button_2.show();

# ... #

box1.show();
window.show();

```

```
gtk.main();
```

لاحظ اننا قمنا بإضافة سطر واحد جديد، مع داله جديد.

```
hello_world_button.connect( 'clicked',  
the_new_window );
```

في هذا السطر قمنا بـ "ربط" الزر الاول بـ "حدث" يُسمى `clicked`.
عندما يقوم المستخدم بالحدث `clicked` يتم استدعاء داله اسمها `the_new_window`، لابد و انك خمنت إن الحدث `clicked` يعني الضغط، هذا يعني عندما يُضغط الزرُ قمْ بإستدعاء الداله `the_new_window`، لنرى ما هي هذه الداله التي عرفناها :

```
def the_new_window( widget, data = None ):  
    new = gtk.Window();  
    new.set_title( "نافذتنا الجديده" );  
  
    msg = gtk.Label( " هذا برنامجي الاول مع "PyGTK"  
);  
    msg.show();  
  
    new.add( msg );  
  
    new.show();
```

من المفترض إنّه لا شئ جديد عليك في هذه الداله، كل ما قمنا به هو إنشاء نافذه جديده و وضعنا لها عنوان "نافذتنا الجديده" و اصفنا إليها نص للقراء "هذا برنامجي الاول مع PyGTK"، الشئ الجديد في هذه الداله هي الاداة `Label` والتي لم نتعامل معها مسبقا، تقوم هذه الداله بطباعة

النص الذي تقوم بوضعه كنص للقراءة.

هناك شيء هام ذكرناه مسبقاً و سوف نعيد ذكره هنا بشيء من الإسهاب، جميع الدوال التي صنفناها من النوع `callback` يجب أن تستقبل بارامترين على الأقل، وهذا الذي تلاحظه عندما عرفنا الداله `the_new_window`، البارامتر الاول و الذي سميناه `widget` يُمثل الاداة التي استدعت الداله `the_new_window`، و في مثالنا هذا الاداة التي قامت بإستدعاء الداله `the_new_window` هي `hello_world_button` (الزر المكتوب عليه مرحبا بالعالم)، في هذه الحاله يكون البارامتر `widget` هو نفسه `hello_world_button`. و بالتالي يمكنك استخدام جميع الدوال التي يقدمها الصنف `gtk.Button` داخل الداله `the_new_window` عن طريق البارامتر `.widget`.

نتقل إلى البارامتر الثاني وهو `data`. تخيل معي اننا نريد تمرير بعض البيانات إلى الداله `the_new_window`، كيف يمكننا ذلك؟ نعود لسطرنا الجديد الذي أضفناه مؤخراً :

```
hello_world_button.connect( 'clicked',  
the_new_window );
```

حتى نقوم بتمرير بعض البيانات إلى الداله `the_new_window` نقوم بتمرير بارامتر ثالث إلى الداله `connect`. مثلاً نريد تمرير كلمة "test" إلى `the_new_window`، يتم ذلك كالتالي :

```
hello_world_button.connect( 'clicked',  
the_new_window, "test" );
```

بالطبع يمكننا تمرير انواع بيانات اخرى، مثلا :

```
info = [ 'One', 'Two', 'Three' ];  
hello_world_button.connect( 'clicked',  
the_new_window, info );
```

الآن كيف يمكننا استقبال البيانات التي قمنا بإرسالها؟ يتم ذلك من خلال البارامتر `data`، اذهب إلى الدالة `the_new_window` وقم بطباعة محتوى البارامتر `data` في نهايتها، وجرّب المثالين السابقين، في اول مثال سوف تطبع كلمة `test` و في المثال الثاني سوف تطبع السطر `['One', 'Two', 'Three']`، بالطبع هذا الاسلوب سيفيدك كثيرا اذا كنت لا تستخدم `OOP` في كتابة برامجك، لانك احيانا تحتاج إلى التحكم في بعض الادوات من داخل دوال `callback`. و لن تتمكن من ذلك إلا اذا مررت الكائنات الخاصه بهذه الادوات إلى دالة `callback`. سوف يمر هذا علينا في امثلتنا القادمه ان شاء الله :-).

هكذا نكون قد انتهينا بفضل الله من شرح المفاهيم الاساسيه، ننتقل الآن إلى المثال الآخر حتى نتدرب اكثر.

المثال الثاني : برنامج تحويل درجة الحرارة

سوف نقوم الآن باستخدام ما تعلمناه و نبني برنامجاً جديداً، يأخذ هذا البرنامج درجة الحرارة بالسيليزي و يحولها إلى الفهرنهايتي، هذا يعني إنَّ برنامجنا يحتوي على مربع نص لأخذ القيمة السيليزيه من المستخدم، و كذلك سوف نحتاج إلى زر يضغط المستخدم عليه عند الانتهاء من وضع القيمة المراد تحويلها، و اخيرا نحتاج إلى نص (Label) لطباعة الناتج، يُفترض أن يكون شكل برنامجنا كالتالي :



لنبدأ أولاً بكتابة

الاسطر التي اتفقنا على كتابتها في بداية الكتاب :

```
# -*- coding: utf-8 -*-
```

```
import pygtk;
pygtk.require( '2.0' );
import gtk;
```

نبني الآن النافذة ونضع عنواناً لها، و نكتب الدالتين اللتان تساعدان على الخروج من البرنامج :

```
def delete_event( widget, data ):
```

```

return False;

def destroy( widget, data = None ):
    gtk.main_quit();

# ... #

window = gtk.Window();

window.set_title( "برنامج التحويل" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

```

كما اتفقنا في البدايه، سوف نحتاج إلى مربع نص، زر و نص مقروء (Label)، سنضعهم اسفل بعضهم البعض، و بالتالي سنستخدم صندوق من نوع VBox، سنقوم بإنشاء هذا الصندوق ثم نضيفه إلى النافذه كما تعلمنا من المثال السابق.

```

main_box = gtk.VBox();
window.add( main_box );

```

لنبدأ بإضافة مربع النص أولاً، لاحظ اننا و للمره الاولى نستخدم هذه الاداة، نقوم بإنشاءها اولاً عن طريق إنشاء كائن لها ثم نضيفها داخل صندوقنا و اخيراً نقوم بإظهارها :

```

text_entry = gtk.Entry();
main_box.pack_start( text_entry );
text_entry.show();

```

الآن نضيف اداة النص و التي تعرفنا عليها مسبقاً و كانت باسم Label، تكون في البدايه فارغه، و سوف يتم تغييرها فيما بعد :

```
status_label = gtk.Label( "" );
main_box.pack_start( status_label );
status_label.show();
```

لننشئ زر و نضع فوقه كلمة "موافق"، ثم نضيفه إلى صندوقنا و اخيراً نظهره :

```
ok_button = gtk.Button( "موافق" );
main_box.pack_start( ok_button );
ok_button.show();
```

و اخيراً سوف نُظهر الصندوق و النافذه الرئيسيه و نستدعي الداله الاساسيه gtk_main :

```
main_box.show();
window.show();
gtk.main();
```

صممنا الآن واجهه بسيطه لبرنامجنا، لاحظ إنّ البرنامج لحد الآن لا يقوم بأي عمل مفيد، حتى يقوم البرنامج بعمل مفيد سنقوم بربط الزر بحدث، بحيث عندما يتم الضغط على الزر تستدعي داله معينه تقوم بالعملية الحساييه و تطبع الناتج، نتوجه إلى السطرين اللذين قمنا بكتابتهما :

```
ok_button = gtk.Button( "موافق" );
```

```
main_box.pack_start( ok_button );
```

و نضيف في اسفلهما دالة الربط. لابد و انك تعرفها (-):

```
widgets = [ status_label, text_entry ];  
ok_button.connect( 'clicked', clickEvent,  
widgets );
```

لنتوقف هنا قليلاً، كما تعلم إنّ البارامتر الاول هو الحدث المطلوب تنفيذ دالة معينة عند حدوثه، و `clicked` بمعنى الضغط. البارامتر الثاني هي الداله التي تستدعى عندما يتم تنفيذ الحدث، وفي حالتنا هذه الحدث هو الضغط على الزر.

البارامتر الثالث وهو ما جعلني اتوقف هنا، هل تذكر شرحنا السابق عن موضوع تمرير البارامترات إلى دوال `callback`؟ هنا قمنا بتمرير مصفوفه تحتوي على الاداتين الموجودتين في برنامجنا كبارامتر للداله `clickEvent` التي سوف نكتبها بعد قليل، و سوف تستقبل `clickEvent` هذا البارامتر باسم `data`. لابد و انك تعرف هذه المعلومات و لكن وجب التذكير، قد تتساءل لما ذا مررنا هذه المصفوفه؟ سيأتيك الجواب عند تعريف الداله `clickEvent`.

لنصعد قليلاً، و بالضبط اسفل الداله `destroy` سوف نعرّف دالتنا `clickEvent`. تقوم هذه الداله بأخذ القيمه الموجوده في مربع النص تضربها في 1.8 ثم تضيف عليه 32 و بالتالي يكون الناتج درجة الحراره بالفهرنهايت، نكتب اولاً رأس الداله و التي تستقبل بارامترين كما شرحنا مسبقاً :

```
def clickEvent( widget, data = None ):
```

يُعتبر البارامتر `data` الآن مصفوفة تحتوي على عنصرين، العنصر الأول هو الكائن `status_label` و الذي يتيح لنا التحكم بأداة النص (`Label`). أما العنصر الثاني هو الكائن `text_entry` و الذي يتيح لنا التحكم في أداة مربع النص، و بالطبع البارامتر `widget` يتيح لنا التحكم بالزر.

مررنا الكائن `status_label` على الدالة `clickEvent` حتى تتمكن من تغيير النص الموجود في أداة النص (`Label`) من داخل الدالة، أما الكائن `text_entry` مررنا حتى تتمكن من أخذ القيمة التي يكتبها المستخدم في مربع النص، هذا يعني إننا نريد الوصول إلى هذه الأدوات من داخل الدالة، و بالتالي مررنا كائنات هذه الأدوات كبارامتر للدالة.

الآن يحتوي الفهرس الأول من المصفوفة `data` و هو `data[0]` على الكائن `status_label` و يحتوي الفهرس الثاني من المصفوفة `data` و هو `data[1]` على الكائن `text_entry` و حتى نقوم بالتسهيل نخزن القيم في متغيرات جديدة ذات اسم اوضح :

```
status_label = data[ 0 ];  
text_entry = data[ 1 ];
```

نأخذ الآن القيمة الموجوده في داخل مربع النص بإستخدام الدالة `get_text` و التي توفرها الفئة `gtk.Entry` :

```
val = text_entry.get_text();
```

نحوّل هذه القيمة من نوع `string` إلى نوع `float` :

```
val = float( val );
```

نُجري عمليتنا الحسابية :

```
result = ( val * 1.8 ) + 32;
```

نحول الناتج من نوع float إلى string مره اخرى من اجل كتابته
داخل اداة النص :

```
result = str( result );
```

اخيراً نغير النص الموجود في اداة النص باستخدام الداله set_text و
التي توفرها الفئة gtk.Label :

```
status_label.set_text( result );
```

بهذا الشكل تكون شيفرة الداله كامله كالتالي :

```
def clickEvent( widget, data = None ):  
    status_label = data[ 0 ];  
    text_entry = data[ 1 ];  
  
    val = text_entry.get_text();  
    val = float( val );  
  
    result = ( val * 1.8 ) + 32;  
    result = str( result );  
  
    status_label.set_text( result );
```

نعود ثانية لنقطة البارامتر data، هناك طريقه اخرى بدلاً من تمرير

الادوات التي نريد التحكم بها من داخل الداله فبدلاً من تمريرها كبارامترات للداله `clickEvent` يمكننا استخدام الكلمة `global` للوصول إلى كائنات هذه الادوات بدون الحاجة إلى تمريرها كبارامترات، وبالتالي تكون دالتنا بهذا الشكل :

```
def clickEvent( widget, data = None ):
    global text_entry, status_label;

    val = text_entry.get_text();
    val = float( val );

    result = ( val * 1.8 ) + 32;
    result = str( result );

    status_label.set_text( result );
```

و السطران :

```
widgets = [ status_label, text_entry ];
ok_button.connect( 'clicked', clickEvent,
widgets );
```

يصبحان بهذا الشكل :

```
ok_button.connect( 'clicked', clickEvent
);
```

بدون تمرير اي شيء، ما رأيك أليست إسهل و اكثر اختصاراً؟ (-) سنستخدم `global` من الآن فصاعداً، لاحظ إنه لا يمكن استخدام الكلمة `global` إلا مع المتغيرات العامة (Global Variables)، أما

المتغيرات المحلية (Local Variables) و التي يتم تعريفها داخل الدوال (غير الدالة الحالية) لا يمكن الوصول إليها من خلال الكلمة global لأنها تختفي بمجرد أن تنهي الدالة عملها.

إنتهينا الآن من الأمور الأساسية في برنامجنا، ما رأيكم الآن بإضافة ميزة تسهل على مستخدم البرنامج؟ :-)

وفقاً للشفيرة التي كتبناها منذ قليل للبرنامج فإنه يجب على المستخدم أن يضغط على زر موافق بعد كتابته للقيمة المطلوبة، ما نريده من برنامجنا الآن هو أن يترك للمستخدم حرية الاختيار إما أن يضغط على زر "موافق" و هذا ما كتبناه في الأعلى، أو أن يضغط على زر الإدخال في لوحة المفاتيح ليتم عرض الناتج و هذا ما سنعمل عليه الآن.

أحد الأحداث التي تقدمها PyGTK هي `key_release_event` يُربط هذا الحدث بعدما يضغط المستخدم على زر معين في لوحة المفاتيح و يرفع إصبعه عنه (أي لا يظل ضاغطاً عليه).

السؤال الآن سنربط الحدث `key_release_event` بأي أداة؟ في حالتنا هذه يمكننا ربطه بالنافذة الرئيسية للبرنامج و كذلك يمكننا ربطه بمربع النص، لك حرية الاختيار و لكنني هنا سأربطه بمربع النص هنا، لاحظ إنه يمكننا ربطه في النافذة الرئيسي هنا لأن لدينا مربع نص واحد فقط في برنامجنا، و لكن إذا كان لدينا أكثر من مربع نص (أو أي أداة قد تستقبل دخل من لوحة المفاتيح) فإن الأمور ستختلف، و بالتالي أنصحك دائماً بربط الحدث مع الأداة التي تخص هذا الحدث.

لنبدأ بكتابة الشيفرة :

فوق سطر إظهار مربع النص :

```
text_entry.show();
```

سنربط مربع النص بالحدث `key_release_event` كالتالي :

```
text_entry.connect( 'key_release_event',  
keyEvent );
```

قبل البدء بكتابة الدالة `keyEvent` هناك نقطة لا بد من توضيحها، بعض الأحداث مثل `key_release_event` تحمل بعض المعلومات، فمثلاً في حالتنا الآن فإن الحدث `key_release_event` يحمل بعض المعلومات التي سنستفيد منها، هذه المعلومات يتم تمريرها كبارامتر ثاني للدالة `Callback` التي سيتم إستدعاؤها عند حدوث الحدث.

لاحظ هنا الإختلاف بين `key_release_event` و حدث الضغط `clicked` الذي إستخدمناه منذ قليل، فالحدث الأول يُمرر بعض المعلومات الخاصة كبارامتر ثاني بينما الحدث الثاني لا يفعل ذلك.

حسناً ما هي هذه المعلومات التي يمررها `key_release_event`؟ في الحقيقة هي عدّة متغيرات موجودة في كائن من نوع `gtk.gdk.Event` و لكن المتغير الذي يهمنا و الذي يمرره الحدث `key_release_event` إلى دالتنا هو `keyval`.

يحتوي المتغير `keyval` على شيفرة المفتاح الذي ضغطه المُستخدم، نحتاج هذا المتغير لأننا سنجعل الناتج يظهر فقط عندما يضغط المستخدم على مفتاح الإدخال، أما المفاتيح الأخرى التي يضغطها المستخدم لا تهتمنا لأنها ستكون فيما بعد القيمة السليزية التي سنحولها إلى فهرنهايته.

لنبدأ بالعمل الحقيقي الآن :-)، نعرف الدالة `keyEvent` كما تعودنا :

```
def keyEvent( widget, data ):
```

البارامتر `data` الآن يحمل كائناً من نوع `gtk.gdk.Event` كما أسلفت، و المتغير `keyval` داخل هذا الكائن هو الذي يحتوي على القيمة المطلوبة و هو رقم المفتاح على لوحة المفاتيح.

سنستخدم دالة تُسمى `keyval_name` تُقدمها مكتبة `GDK`. وظيفة هذه الدالة هي إعادة إسم المفتاح. سنستخدمها لأن المتغير `keyval` يحتوي على رقم المفتاح، و بالتالي نحن بحاجة لإسمه حتى نتمكن من تحديد إن كان المفتاح المضغوط هو مفتاح الإدخال أم لا.

السطر التالي يُخزّن إسم المفتاح المضغوط في متغير جديد إسمه `key` :

```
key = gtk.gdk.keyval_name( data.keyval );
```

إذا كان مفتاح الإدخال هو ما ضغط عليه المستخدم فستكون قيمة `key` هي `Return`. و بالتالي كل ما علينا فعله هو إستدعاء الدالة القديمة التي كتبناها و هي `clickEvent` لانها هي التي تقوم بالحسابات كما تعلم :

```
if ( key == 'Return' ):  
    clickEvent( None );
```

جرب الآن تشغيل البرنامج، و أكتب قيمه معينه ثم إضغط على مفتاح الإدخال في لوحة مفاتيحك بدلا من الضغط على زر موافق في البرنامج، و

سيظهر الناتج (-):

الشفيرة الكاملة :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

def clickEvent( widget, data = None ):
    global text_entry, status_label;

    val = text_entry.get_text();
    val = float( val );

    result = ( val * 1.8 ) + 32;
    result = str( result );

    status_label.set_text( result );

def keyEvent( widget, data ):
    key =
```

```

gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        clickEvent( None );

# ... #

window = gtk.Window();

window.set_title( "برنامج التحويل" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

# ... #

main_box = gtk.VBox();
window.add( main_box );

# ... #

text_entry = gtk.Entry();
main_box.pack_start( text_entry );
text_entry.connect( 'key_release_event',
keyEvent );
text_entry.show();

# ... #

status_label = gtk.Label( "" );
main_box.pack_start( status_label );
status_label.show();

```

```
# ... #  
  
ok_button = gtk.Button( "موافق" );  
main_box.pack_start( ok_button );  
ok_button.connect( 'clicked', clickEvent  
);  
ok_button.show();  
  
# ... #  
  
main_box.show();  
  
window.show();  
  
gtk.main();
```

المثال الثالث : آله حاسبه

قبل الانتقال إلى الموضوع التالي لنأخذ مثلاً آخرًا، هذا المثال عبارته عن آله حاسبه، تقوم بالعمليات الاساسيه، جمع، طرح، ضرب، قسمه.

تحتوي واجهة البرنامج على ثلاث مربعات نص يكتب في المربع الاول العدد الاول و تكتب العمليه المطلوب تنفيذها في المربع الثاني و اخيرا يكتب الرقم الثاني في المربع الاخير، بالاضافه إلى ذلك يكون هناك زرا لإظهار الناتج و يكون هناك نصا (Label) يظهر فيه الناتج بعد الضغط على الزر، شكل البرنامج كالتالي :



كما تعودنا نكتب اولاً الشيفرات الاساسيه و التي دائماً ما نستخدمها في بداية برامجنا :

```
# -*- coding: utf-8 -*-
```

```
import pygtk;
pygtk.require( '2.0' );
import gtk;
```

نبني الآن نافذه البرنامج مع الدالتين اللتين تعودنا دائماً ربط النوافذ بهما :

```
def delete( widget, data ):
```



```

return False;

def des( widget, data = None ):
    gtk.main_quit();

win = gtk.Window();

win.set_title( "آله حاسبه" );
win.connect( 'delete_event', delete );
win.connect( 'destroy', des );

```

لنأخذ وقتنا بالتفكير بعدما بنينا النافذة. الصندوق الرئيسي الذي سنضعه هل يكون من نوع HBox ام VBox؟

كما اتفقنا سيكون هنالك ثلاث مربعات نص، وجميعها بجانب بعضها البعض، و في اسفلها جميعها يكون هناك زر و اسفل الزر هناك نص تظهر فيه النتائج، في هذه الحالة نحتاج إلى النوعين، نحتاج الصندوق HBox من اجل مربعات النص الثلاثه بجانب بعضهن، و نحتاج إلى صندوق VBox حتى نضع الزر اسفل هذه الادوات و نضع النص اسفل المربع.

الصندوق الرئيسي سيكون من نوع VBox، نقسم هذا الصندوق إلى ثلاث خانات عموديه، الخانه الاولى تحتوي على مربعات النص، الخانه الثانيه تحتوي على الزر و الخانه الثالثه تحتوي على النص (Label). المشكله هنا أنّ الخانَه الواحده لا تستوعب إلا اداة واحده، و بما أنّ نوع الصندوق VBox اذا سيتم وضع كل اداة جديده يتم اضافتها إلى الصندوق في اسفل الاداة التي تسبقها.

إذا لدينا مشكله في الخانه الاولى و هي كيف نضع الثلاث ادوات بجانب

بعضهن البعض؟ سنقوم بإنشاء صندوق جديد من نوع HBox غير الصندوق الرئيسي، بعدها نضيف مربعات النص إلى الصندوق الثاني HBox و أخيرا نضيف الصندوق HBox إلى الصندوق الرئيسي من نوع VBox في الخانه الاولى، هذه الطريقة هي الطريقة المُستخدمه دائما لحل مثل هذه المشاكل، عندما يكون لدينا واجهه معقده و تحتوي على ادوات بجانب بعضها و ادوات اخرى اسفلها نستخدم الحل المشروح و الذي نضع عليه مثال عملي الآن.

حسناً لنبدأ، نضيف الآن الصندوق الرئيسي و الذي كما اتفقنا سيكون من نوع VBox :

```
main_box = gtk.VBox();  
win.add( main_box );
```

كما إتفقنا، في الخانه الاولى سنضع مربعاتِ النصِ الثلاثه بجانب بعضهن، ووفقا لما شرحناه بالاعلى سننشئ صندوقا جديدا من نوع HBox من اجل تنفيذ هذا الغرض :

```
box1 = gtk.HBox();
```

نضيف الآن مربع النص الاول بشكل عادي و كما تعلمنا في السابق، و لكننا بدلا من اضافته إلى الصندوق الرئيسي main_box نضيفه إلى الصندوق الثاني box1، يُستخدم مربع النص هذا من اجل كتابة الرقم الاول :

```
first_number = gtk.Entry();  
box1.pack_start( first_number );  
first_number.show();
```

نضيف بعدها المربع الثاني، و الذي يُستخدم من اجل كتابة نوع العمليه المطلوبه + او - او * او \

```
operation = gtk.Entry();
box1.pack_start( operation );
operation.show();
```

و اخيرا نضيف المربع الثالث و الذي يُستخدم لكتابة الرقم الثاني :

```
second_number = gtk.Entry();
box1.pack_start( second_number );
second_number.show();
```

الآن و بعد اضافة جميع مربعات النص إلى الصندوق الثاني، نضيف الصندوق إلى الصندوق الرئيسي ثم نظهره :

```
main_box.pack_start( box1 );
box1.show();
```

نضيف الآن بقية الادوات بشكل عادي طالما انها سوف تكون تحت بعضها البعض، بالطبع لا داعي ان اقول لك اننا سوف نضيفها إلى الصندوق الرئيسي (-:)، نضيف اولا الزر :

```
button = gtk.Button( "أظهر الناتج" );
main_box.pack_start( button );
button.show();
```

بعد الزر نضيف النص :

```
result_label = gtk.Label();
main_box.pack_start( result_label );
result_label.show();
```

و أخيراً نَظْهر الصندوق الرئيسي، النافذه، و ننادي الداله الاساسيه :

```
main_box.show();
win.show();
gtk.main();
```

انتهينا الآن من تصميم الواجهه فحسب، يجب ان يكون برنامجنا اكثر من مجرد واجهه :-)، يبدأ عمل البرنامج عندما يضغط المستخدم على الزر، يستدعي الزر داله معينه تقوم بالعملية الرياضيه ثم تطبع الناتج على النص (Label) هذا كل شيء.

اذا نبدأ مع الزر، نربط الزر بالحدث `clicked` ليتم استدعاء داله اسمها `doMath` عن طريق السطر التالي و الذي سوفه نضيفه فوق السطر الذي يضع الزر في الصندوق الرئيسي :

```
button.connect( 'clicked', doMath );
```

نُعرف الآن الداله `doMath` في اسفل الدوال التي عرفناها مسبقاً :

```
def doMath( widget, data = None ):
```

نستخدم `global` كما شرحنا مسبقاً حتى نتمكن من التحكم بالادوات من

داخل الداله :

```
global result_label, first_number,  
second_number, operation;
```

نبدأ اولاً في التحقق اذا كان المستخدم ترك مربعات النص فارغه ام لا،
في حال ترك احدها فارغه نظهر له رسالة خطأ :

```
operand1 = first_number.get_text();  
operand2 = second_number.get_text();  
op = operation.get_text();
```

```
if ( not operand1 or not operand2 or not  
op ):  
    result_label.set_text( " يرجى تعبئة "  
المعلومات المطلوبه " );
```

اذا كان كل شئ يعمل حسب الاصول و جميع المعلومات المطلوبه
متوفره، نبدأ بالعملية الحسابيه، في بادئ الامر نحول نوع كل من المتغيرين
operand1 و operand2 إلى integer حتى يكون ناتج العملية
الحسابيه صحيحاً لأنها حالياً عبارته عن نصوص (أو سلاسل Strings)
:

```
else:  
    operand1 = int( operand1 );  
    operand2 = int( operand2 );
```

لاحظ إنّ else هنا تابعة للجملة الشرطية السابقة و التي تتحقق من كتابة
المستخدم للقيم المطلوبه منه.

نتحقق الآن من النص الذي ادخله المستخدم في المربع الثاني، هل هو + او - او * او \ ثم نقوم بالعملية المطلوبه وفقا لمحتوى المربع الثاني، اذا كان محتوى المربع الثاني لا يساوي ايا من القيم المذكوره سنطبع رساله خطأ للمستخدم و نخرج من دالة doMath :

```
result = operand1;

if op == "+":
    result += operand2;
elif op == "-":
    result -= operand2;
elif op == "*":
    result *= operand2;
elif op == "/":
    result /= operand2;
else:
    result_label.set_text( " العملية التي قمت
    بإختيارها غير صحيحة ");
    return False;
```

و اخيراً نطبع الناتج للمستخدم بعد تحويله إلى نص :

```
result_label.set_text( str( result ) );
```

آلئنا الحاسبة البسيطة كاملة الآن :-)، و لكن ما رأيك لو أضفنا ميزة مفتاح الإدخال التي أضفناها في برنامج تحويل الحرارة؟ يُدخل المستخدم القيم المطلوبة ثم يضغط على مفتاح الإدخال في لوحة مفاتيحة ليظهر الناتج بدلا من إجباره على الضغط على زر "أظهر الناتج" في البرنامج.

كما تعلمنا سابقاً، يجب ربط إحدى الأدوات بالحدث `key_release_event`. و لكن السؤال الآن ما هي الأداة التي سنربط " حدث تحرير المفتاح " فيها؟ هل هو مربع النص الأول أم الثاني أم الثالث؟

في الحقيقة يجب علينا ربط الحدث لمربعات النص الثلاثة، لماذا؟ لأننا لا نعلم أين قد يضغط المستخدم على مفتاح الإدخال، غالباً سيضغط عليه بعدما يكتب القيمة في المربع الثالث لأنها آخر قيمة سيدخلها، لكن ماذا لو إكتشف إنه كتب قيمة خاطئة في المربع الأول و ذهب لتعديلها؟ سيضغط على مفتاح الإدخال بعدما يكتب القيمة في المربع الأول، و إن كان الحدث مربوط بالمربع الثالث فقط فلن يعلم المستخدم ذلك و سيضطر إلى الضغط على زر " أظهر الناتج " عن طريق الفأرة و هكذا إختفت الميزة التي أضفناها! :-)

لنبدأ بربط الأحداث، نضيف الأسطر التالية قبل سطر إضافة الصندوق الثاني في الصندوق الرئيسي :

```
first_number.connect( 'key_release_event'
, keyEvent );
operation.connect( 'key_release_event',
keyEvent );
second_number.connect( 'key_release_event'
, keyEvent );
```

الدالة `keyEvent` تعمل بنفس الطريقة التي كتبناها في برنامج تحويل درجة الحرارة، كالتالي :

```
def keyEvent( widget, data ):
```

```

        key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        doMath( None );

```

الشفيرة الكاملة :

```

# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def doMath( widget, data = None ):
    global result_label, first_number,
second_number, operation;

    operand1 = first_number.get_text();
    operand2 = second_number.get_text();
    op = operation.get_text();

    if ( not operand1 or not operand2
or not op ):

```



```

        result_label.set_text( " يرجى تعبئة "
المعلومات المطلوبه " );
    else:
        operand1 = int( operand1 );
        operand2 = int( operand2 );

        result = operand1;

        if op == "+":
            result += operand2;
        elif op == "-":
            result -= operand2;
        elif op == "*":
            result *= operand2;
        elif op == "/":
            result /= operand2;
        else:
            result_label.set_text( "
العملية التي قمت بإختيارها غير صحيحة " );
            return False;

        result_label.set_text(
str( result ) );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        doMath( None );

# ... #

```

```

win = gtk.Window();

win.set_title( "آله حاسبه" );
win.connect( 'delete_event', delete );
win.connect( 'destroy', des );

# ... #

main_box = gtk.VBox();
win.add( main_box );

# ... #

box1 = gtk.HBox();

# ... #

first_number = gtk.Entry();
box1.pack_start( first_number );
first_number.show();

# ... #

operation = gtk.Entry();
box1.pack_start( operation );
operation.show();

# ... #

second_number = gtk.Entry();
box1.pack_start( second_number );
second_number.show();

```

```

# ... #

first_number.connect( 'key_release_event'
, keyEvent );
operation.connect( 'key_release_event',
keyEvent );
second_number.connect( 'key_release_event'
', keyEvent );

# ... #

main_box.pack_start( box1 );

box1.show();

# ... #

button = gtk.Button( "أظهر الناتج" );
button.connect( 'clicked', doMath );
main_box.pack_start( button );
button.show();

# ... #

result_label = gtk.Label();
main_box.pack_start( result_label );
result_label.show();

# ... #

main_box.show();

win.show();

```

```
gtk.main();
```

مدخل إلى Glade

قد يعتبر البعض كتابة الشيفره المصدريه الخاصه بواجهه البرنامج عمليه ممله و على العكس قد يكون هناك اشخاص يتمتعون عندما يكتبون شيفرة واجهه البرنامج، و لكن لنفكر بالموضوع قليلا، اذا قررنا كتابة برنامج ضخم باستخدام PyGTK فإننا سنستغرق الكثير من الوقت من اجل كتابة الشيفره المصدريه الخاصه بالواجهه، و قد تكون عمليه الاضافه على الواجهه في ما بعد عمليه متعبه و طويله، لحسن الحظ هناك حل لهذه المشكله و هو برنامج Glade.

يسهل عليك برنامج Glade تصميم واجهات برنامجك، فبدلاً من كتابة شيفرة الواجهه تقوم بتصميم الواجهه بشكل مرئي عن طريق برنامج Glade، سوف يفهم مبرمجي Gambas و Delphi و Visual Basic قصدي، يقوم برنامج Glade بعدها بتوليد ملفات من نوع xml تستخدمها فيما بعد داخل شيفرة برنامجك حتى تتمكن من التحكم في واجهتك الرسوميه.

إلى وقت إصدار الطبعة الثانية من هذا الكتاب (سنة 2013) فإنّ الإصدارات الحالية من Glade هي الثالثة، و أحدث إصدار هو 3.12، الجيل الثالث من Glade كان اعادة كتابه من الصفر، من وجهة نظري ان واجهه الجيل الثالث افضل بكثير من واجهه الجيل الثاني.

يمكن التعامل مع ملف XML الناتج من برنامج Glade في النص البرمجي كما سنرى بعد قليل، في الطبعة الأولى (سنة 2009) من هذا الكتاب تم شرح المكتبة البرمجية Libglade و هي المسؤولة عن التعامل مع ملفات XML الناتجة عن برنامج Glade في النص البرمجي، في عام 2007 صدرت النسخة 2.12 من مكتبة GTK و جاءت معها

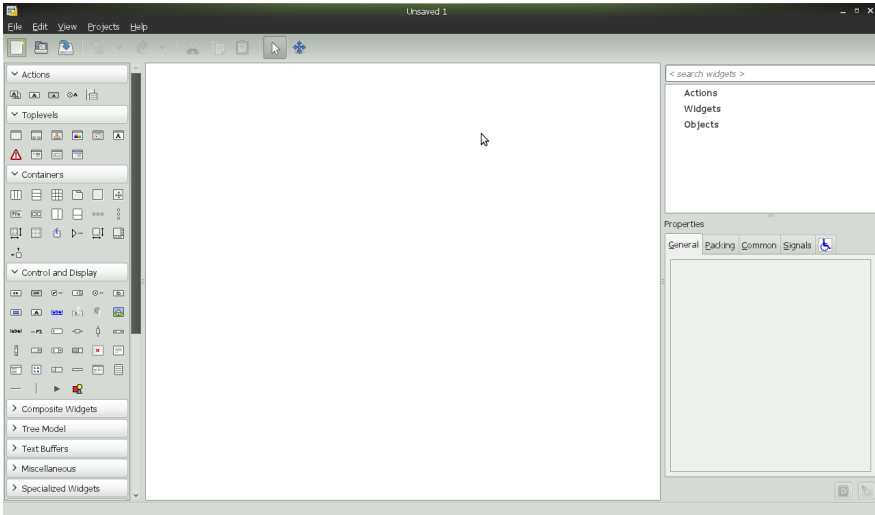
مكتبة `GtkBuilder`، و لها نفس وظيفة `Libglade`، و لكنّها جاءت لتكون بديل عن `Libglade` حيث تحتوي على ميزات ليست موجودة في `Libglade`.

في الوقت الحالي (2013) بدأ المبرمجون بالانتقال من `Libglade` إلى `GtkBuilder`، و بالتالي سيكون الشرح القادم قائم على مكتبة `GtkBuilder` بدلا من `Libglade`.

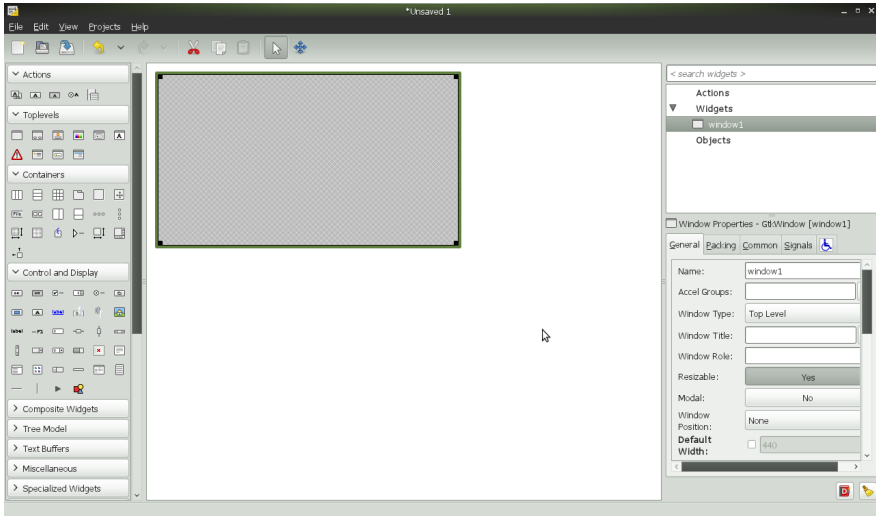
سوف نتعلم في الصفحات القادمة إن شاء الله كيفية استخدام `Glade` و استخدام `GtkBuilder` لإنشاء تطبيقاتنا.

التصميم مع Glade

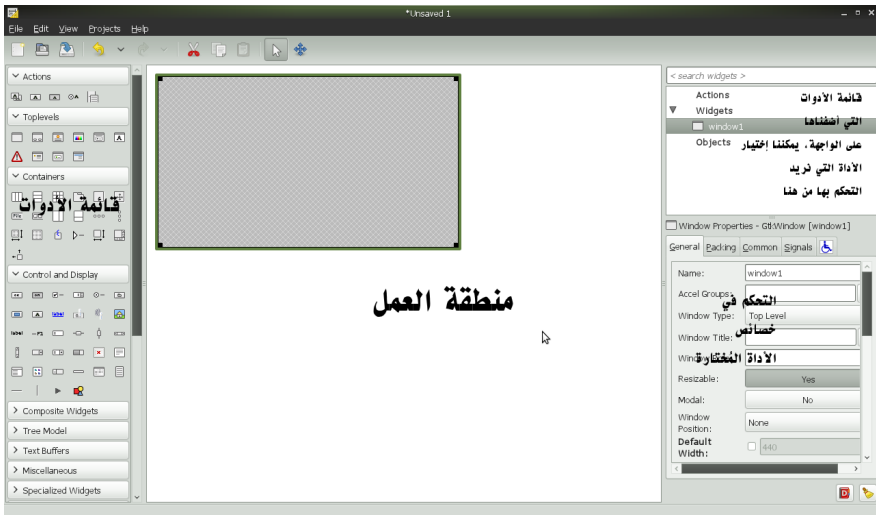
لقد تعلمنا في الاقسام السابقة كيف نكتب شيفرة واجهة المستخدم، عملية تصميم واجهة المستخدم مع Glade اسهل بكثير، بمجرد رؤيتك لواجهة البرنامج سوف ترى إنّ العملية بسيطة، لنلقي نظره على الصورة التاليه :



هذا هو برنامج Glade الذي سنستخدمه لتصميم واجهاتنا من الآن فصاعداً، حسنا لنرى الآن كيف يمكننا استخدام هذا البرنامج في تصميم واجهتنا، نلاحظ على الجانب الايسر قائمه تحتوي على الادوات مثل النافذه و الصناديق و مربعات النص و غيرها الكثير من الادوات التي لم نستخدمها مسبقا، هذه هي جميع الادوات التي تقدمها GTK، كما تعودنا دائما في البدايه نحن بحاجة إلى نافذه حتى نضيف عليها ادوات البرنامج، ستجد النافذه في قائمة Toplevels و اسمها Window، الصورة التاليه توضح لك اننا اضفنا نافذه إلى واجهتنا التي نبنها :

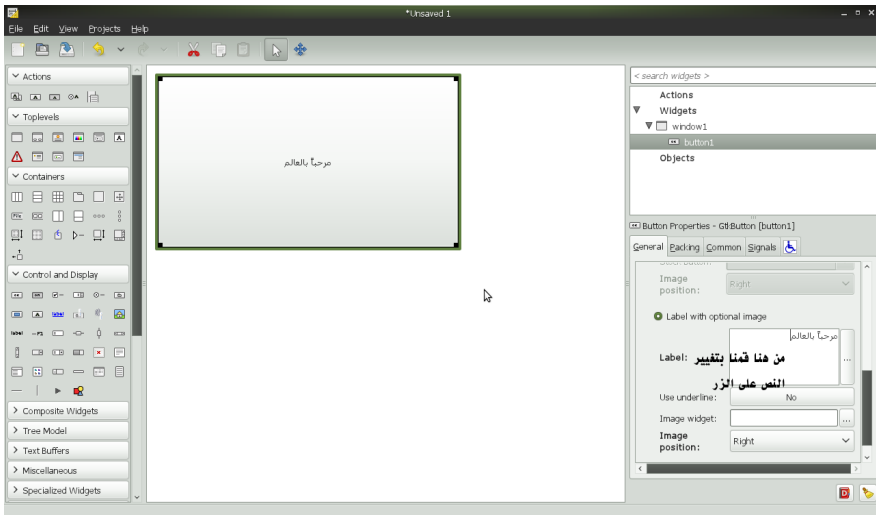


لاحظ الفرق بعد اضافة النافذه، في المنتصف ظهرت النافذه التي اضفناها و التي سوف نضيف فوقها الادوات، اذا الجزء الذي يقع في المنتصف هو مكان العمل، اما جهة اليمين نلاحظ ان هناك قائمه في الاعلى، هذه القائمه تحتوي على جميع الادوات التي اضفناها في مشروعنا هذا، اما اسفل هذه القائمه نجد صندوقا يحتوي على مجموعه من مربعات النص و القوائم و التبويبات، من خلال هذا الصندوق يمكننا التحكم بخصائص اي اداة، و في هذه الصوره بالذات فإننا نتحكم بخصائص النافذه التي اضفناها، الصوره التاليه توضح شرحنا هذا بشكل اوضح :



ما رأيك الآن لو صممنا واجهه بسيطه جداً نتعلم من خلالها كيفية التصميم؟ نريد واجهه تحتوي على زر مكتوب عليه "مرحبا بالعالم" هذا كل شيء!

هيا لنبدأ نضيف اولاً النافذه تجدها كما ذكرنا تحت القسم Toplevels باسم Window، لأننا سنضيف اداة واحده فقط و بالتالي لسنا بحاجة إلى استخدام الصناديق، نضيف الزر إلى النافذه مباشرة، بعد إضافة الزر نذهب إلى خصائصه لنغيرها (يمين الشاشة في الاسفل)، يجب ان يكون التبويت المُختار هو General نجد حقل اسمه label و بجانبه مربع نص كبير، نغير النص في هذا المربع إلى "مرحبا بالعالم"، يجب ان يظهر لك شئ مشابه للصورة التاليه (بالمناسبه لا تعلق بشأن حجم النافذه):



نخزن الآن واجهتنا البسيطة، لاحظ إنك عندما تطلب من Glade تخزين الملف، سيظهر لك في أسفل شاشة التخزين خيارين هما GtkBuilder و Libglade. دائما إختار GtkBuilder فكما وضحت مسبقا فإننا سنستخدم مكتبة GtkBuilder البرمجية من أجل التعامل مع ملف XML الناتج من Glade.

من اجل الترتيب من الافضل إنشاء مجلد خاص ببرنامجنا لنخزن فيه الملف، لنسمي ملفنا باسم gui، بعد التخزين يجب ان يكون لدينا في المجلد ملف باسم gui.glade. هذا الملف هو الواجهه التي صممناها و التي سوف نستخدمها في النص البرمجي لإظهار الواجهه، ننتقل إلى القسم التالي حتى نرى كيف نتعامل مع ملفات Glade في النص البرمجي باستخدام GtkBuilder.

استخدام ما تم تصميمه مع Glade في النص البرمجي

نبدأ برنامجنا كما تعودنا دائماً بإرفاق المكتبات التي سوف نستخدمها :

```
# -*- coding: utf-8 -*-  
  
import pygtk;  
pygtk.require( '2.0' );  
import gtk;
```

نضيف الدوال التي اعتدنا على اضافتها في كل برنامج :

```
def delete_event( widget, data ):  
    return False;  
  
def destroy( widget, data = None ):  
    gtk.main_quit();
```

لنبدأ عملنا الآن مع `GtkBuilder`. في البداية نحن بحاجة إلى إنشاء كائن من `GtkBuilder` حتى تتمكن من تحميل ملف `Glade` و التعامل معه في النص البرمجي :

```
builder = gtk.Builder();
```

سيكون تعاملنا الآن مع عناصر الواجهة الرسومية من خلال الكائن `builder`، بعد خطوة إنشاء الكائن سنستخدم الدالة `add_from_file` و التي يُقدمها لنا `GtkBuilder` من أجل تحميل ملف `Glade` الذي

صممناه، تأخذ هذه الدالة بارامتراً واحداً و هو إسم ملف Glade المراد تحميله، في حالتنا فقد سميناه `gui.glade` :

```
builder.add_from_file( 'gui.glade' );
```

تحتوي واجهتنا هذه على أداتين، الاداة الاولى هي النافذه الرئيسيه، الاداة الثانيه هي الزر المكتوب فوقه "مرحبا بالعالم".

في السابق عندما كنا نكتب واجهاتنا الرسومية عن طريق الاسطر البرمجيّه كان هناك متغير (يحتوي على كائن) لكل أداة نضيفها، و من خلال هذا الكائن (Object) نتحكم في الاداة، لحسن الحظ يمكننا فعل ذلك مع مكتبة `GtkBuilder` عن طريق استخدام الداله `get_object`، تأخذ هذه الداله اسم الاداة التي تريد التحكم بها كبارامتر.

ارجع إلى برنامج `Glade` و شغل الملف `gui.glade`. اختر النافذه و انظر في خصائصها، انظر الخاصيه `Name` سوف تجد ان المكتوب في المربع الواقع بجانب `Name` هو `window1` هذا هو الاسم الذي سوف نمرره إلى الداله `get_object`. كذلك بالنسبه للزر سوف تجد ان اسمه `button1`، بالطبع يمكنك تغيير هذه الاسماء إلى اسماء انسبٍ من وجهة نظري الإبقاء على الاسماء الافتراضيه التي يضعها `Glade` تلقائياً مثل `window1` و `button1` و `button2` عادة سيئة و سوف يظهر ذلك بوضوح عندما تصمم برنامج ذو واجهه معقده، لذلك من الافضل وضع إسم ذو معنى واضح لكل اداه حيث يُبين هذا الإسم وظيفة الأداة في الواجهة الرسومية.

غير الآن اسم النافذه إلى `main_window`. و اسم الزر إلى `hello_world_button` و احفظ التعديلات. حان الوقت لنستخدم

الدالة (-: get_object :

```
window = builder.get_object(  
    'main_window' );  
button = builder.get_object(  
    'hello_world_button' );
```

نقوم بإظهار الادوات عن طريق الدالة show :

```
window.show();  
button.show();
```

نستدعي الدالة الاساسيه :

```
gtk.main();
```

حسناً لا بد و انك تعلم اننا الآن يمكننا استخدام الكائنين window و button من اجل التحكم بالادواتين و بالتالي يمكننا استخدام جميع الدوال التي تقدمها الفئة gtk.Window او الفئة gtk.Button، مثلاً اذا اردنا تغيير عنوان النافذه يمكننا استخدام الداله set_title و هكذا.

الآن اصبح من الواضح كيفية استخدام الملفات التي يولدها برنامج Glade في داخل برامجنا، سوف ننتقل إلى القسم التالي و نتعلم كيف يمكننا التعامل مع الاشارات.

الشيقره كامله :

```
# -*- coding: utf-8 -*-
```

```
import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    False;

def destroy( widget, data = None ):
    gtk.main_quit();

# ... #

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

# ... #

window = builder.get_object(
    'main_window' );
button = builder.get_object(
    'hello_world_button' );

# ... #

window.show();
button.show();

gtk.main();
```

استخدام الاشارات مع ما تم تصميمه مع Glade

هناك طريقتان يمكننا استخدام احدهما للتعامل مع الاشارات عند استخدامنا لـ `GtkBuilder`، الطريقة الاولى هي الطريق التي اعتدنا عليها و استخدمناها في السابق، درسنا قبل قليل الداله `get_object` و التي تسمح لنا باستخدام اي اداة بشكل عادي و كأننا اضفناها عن طريق شيفره برمجيه، و بالتالي يمكننا استخدام الطريقه القديمه و هي الداله `.connect`.

نعود لمثالنا السابق، نريد ربط الزر بحدث الضغط، عندما يضغط المستخدم على الزر يتغير النص الموجود على الزر إلى "تم الضغط على الزر" و بالطبع لا ننسى ربط النافذه بداله الإغلاق :

اولاً نبدأ بدوال الاغلاق :

```
window.connect( 'delete_event',  
delete_event );  
window.connect( 'destroy', destroy )
```

الآن نعمل مع الزر، و نربطه بحدث الضغط كما تعلمنا مسبقاً :

```
button.connect( 'clicked', buttonClicked  
);
```

نكتب الآن الداله `buttonClicked` اعلى الملف :

```
def buttonClicked( widget, data = None ):
    widget.set_label( 'تم الضغط على الزر' );
```

هذا كل شيء!

الشيء كله كامله :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;
import gtk.glade;

# ... #

def delete_event( widget, data ):
    False;

def destroy( widget, data = None ):
    gtk.main_quit();

def buttonClicked( widget, data = None ):
    widget.set_label( 'تم الضغط على الزر' );

# ... #

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

# ... #
```



```

window = builder.get_object(
    'main_window' );
button = builder.get_object(
    'hello_world_button' );

# ... #

window.connect( 'delete_event',
    delete_event );
window.connect( 'destroy', destroy );

button.connect( 'clicked', buttonClicked
);

# ... #

window.show();
button.show();

gtk.main();

```

بالطبع لا داعي للإسهاب عن ما قمنا به لاننا تحدثنا بشكل مفصّل عنه بالاقسام السابقة و يفترض بك أن تكون قد فهمته بشكل جيد.

ننتقل الآن إلى الطريقه الثانيه و هذه الطريقه خاصه بـ Glade. شغل الملف gui.glade مع برنامج Glade مره اخرى، و اختر الزر من اجل تغيير خصائصه، اذهب إلى الخصائص، و هذه المره بدلا من اختيار التبويت General اختر التبويت Signals، من خلال هذا التبويت يمكننا ربط الزر بأي حدث مطلوب.

حالياً الحدث المطلوب ربطه بالزر الموجود في برنامجنا هو `clicked`.
ستجد الحدث `clicked` ضمن قائمة `GtkButton`.

بجانب الكلمة `clicked` سوف نجد مربع نص يمكننا الكتابه فيه، مكتوب اعلى هذا المربع `Handler`، نكتب دائماً في هذا المربع اسما مميزاً، شخصياً افضل كتابة اسم الداله التي سوف تعالج الحدث، يمكننا تغيير الاسم إلى اي شئ آخر و لكن حتى لا نزيد الاسماء في برنامجنا نستخدم اسما موحداً، سنستخدم هذا الاسم في النص البرمجي بحيث نضعه في مصفوفه (او قاموس) و يكون الاسم مفتاح المدخل و تكون قيمته هي الداله التي يجب استدعاءها، في حالتنا هذه الداله التي سوف تعالج الحدث هي `buttonClicked` و التي كتبناها منذ قليل، اذا قم بوضع `buttonClicked` في `Handler`، احفظ المشروع و لننتقل إلى النص البرمجي.

سوف نعدّل على المثال السابق و نحوّل الاشارات فيه من الطريقه العاديه إلى طريقه `Glade`. و بالتالي يجب علينا حذف الأ سطر التاليه من النص البرمجي :

```
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

button.connect( 'clicked', buttonClicked
);
```

نشئ الآن المصفوفه التي اتفقنا على إنشائها، تحتوي هذه المصفوفه على مدخل واحد فقط، له مفتاح باسم `buttonClicked`، و قيمة هذا المفتاح هي الداله `buttonClicked`. لا تخط بين الاثنتين، المفتاح هو

الاسم الذي حددناه في Glade و ليس من الضروري ان يكون بنفس اسم الداله، اما الاسم الثاني فهو اسم الداله نفسها، سوف نسمي هذه المصفوفه باسم signals :

```
signals = { "buttonClicked" :  
buttonClicked }
```

الآن حتى تتم عملية الربط بنجاح نقوم بإستدعاء الداله connect_signals و التي تقدّمها لنا مكتبة GtkBuilder كالتالي :

```
builder.connect_signals( signals );
```

الآن ربطنا الزر بطريقة Glade، يمكنك تجربة البرنامج و سوف يعمل كما كان.

نُكمل عملية الربط الآن و نربط النافذه بحدث الإغلاق حتى نتمكن من إغلاق البرنامج بشكل صحيح، افتح الملف gui.glade مع برنامج Glade مره اخرى، هذه المره اختر النافذه الرئيسيّه، انتقل إلى التبويب Signals.

كما تعودنا سوف نربط النافذه بالحدثين delete_event والذي تجده في GtkWidget، كما اتفقنا سوف نضع اسم ال Handler بنفس اسم الداله و دالتنا هنا هي delete_event. الحدث الثاني هو destroy تجده في GObject، سوف نسمي ال Handler هنا باسم destroy، احفظ العمل و انتقل إلى النص البرمجي.

نضيف مُدخلين جديدين إلى مصفوفة الاشارات و بالطبع هذان المُدخلان يخصّان الحدثين الذين اضعناهما قبل قليل، و بالتالي تصبح مصفوفتنا بعد

التعديل كالتالي :

```
signals = { "buttonClicked" :  
buttonClicked,  
            "destroy" : destroy,  
            "delete_event" :  
delete_event}  
  
builder.connect_signals( signals );
```

جرب البرنامج الآن و ستجده يعمل وفقاً للاصول (-:

الشيء كامله :

```
# -*- coding: utf-8 -*-  
  
import pygtk;  
pygtk.require( '2.0' );  
import gtk;  
import gtk.glade;  
  
# ... #  
  
def delete_event( widget, data ):  
    False;  
  
def destroy( widget, data = None ):  
    gtk.main_quit();  
  
def buttonClicked( widget, data = None ):  
    widget.set_label( 'تم الضغط على الزر' );
```

```

# ... #

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

# ... #

window = builder.get_object(
    'main_window' );
button = builder.get_object(
    'hello_world_button' );

# ... #

signals = { "buttonClicked" :
buttonClicked,
            "destroy" : destroy,
            "delete_event" :
delete_event}

builder.connect_signals( signals );

# ... #

window.show();
button.show();

gtk.main();

```

نقل برنامج تحويل درجة الحرارة إلى Glade

بالطبع تتذكر المثال الثاني الذي كتبناه، و كما تعلم فإننا كتبنا واجهته برمجيا و لم نصممها مع Glade، سوف ننقل واجهة هذا البرنامج الآن إلى Glade، و بالطبع سوف نستند إلى نفس النص البرمجي، النص البرمجي هو :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

def clickEvent( widget, data = None ):
    global text_entry, status_label;

    val = text_entry.get_text();
    val = float( val );

    result = ( val * 1.8 ) + 32;
    result = str( result );
```

```

        status_label.set_text( result );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        clickEvent( None );

# ... #

window = gtk.Window();

window.set_title( "برنامج التحويل" );
window.connect( 'delete_event',
delete_event );
window.connect( 'destroy', destroy );

# ... #

main_box = gtk.VBox();
window.add( main_box );

# ... #

text_entry = gtk.Entry();
main_box.pack_start( text_entry );
text_entry.connect( 'key_release_event',
keyEvent );
text_entry.show();

# ... #

```

```

status_label = gtk.Label( "" );
main_box.pack_start( status_label );
status_label.show();

# ... #

ok_button = gtk.Button( "موافق" );
main_box.pack_start( ok_button );
ok_button.connect( 'clicked', clickEvent
);
ok_button.show();

# ... #

main_box.show();

window.show();

gtk.main();

```

و بعد حذف شيفرة الواجهه الرسوميه يصبح كالتالي :

```

# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):

```



```

    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

def clickEvent( widget, data = None ):
    global text_entry, status_label;

    val = text_entry.get_text();
    val = float( val );

    result = ( val * 1.8 ) + 32;
    result = str( result );

    status_label.set_text( result );

def keyEvent( widget, data ):
    key =
    gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        clickEvent( None );

gtk.main();

```

نبدأ الآن بالتصميم باستخدام Glade، برنامج تحويل الحرارة يستخدم الصناديق، و يستخدم صندوق من نوع VBox و بالتالي أول شيء نفعله بعد إضافة النافذة الرئيسيه هو اضافة صندوق من نوع VBox إلى النافذة، ستجد الصندوق VBox تحت التصنيف Containers تحت اسم .Vertical Box

عندما تضع الصندوق على النافذة سيسألك **Glade** : كم عدد العناصر المطلوبة؟ و كما تعلم إننا نريد إضافة ثلاث أدوات واحده أسفل الاخرى، وبالتالي سوف نحتاج إلى 3 عناصر، بعد اختيارك للعدد 3 و الضغط على موافق تجد إن النافذة تقسمت إلى 3 اقسام.

في القسم الاول نضيف مربع نص تجده باسم **Text Entry** في القسم **Control And Display**، القسم الثاني نضيف نص (**Label**) والذي تجده باسم **Label** في نفس قسم مربع النص، و اخيرا نضيف الزر في القسم الثالث، نعدّل الآن بعض الخصائص :

نختار الزر و نذهب إلى خصائصه، و نغيّر قيمة الخاصية **Label** إلى "موافق"، بعدها نختار النص (**Label**) و نذهب إلى خصائصه و نحذف قيمة الخاصية **Label**. نختار الآن النافذة الرئيسية و نذهب إلى خصائصها و نضع قيمة الخاصية **Window Title** كالتالي : "برنامج التحويل".

نقوم الآن بعمليات الربط، لربط النافذة الرئيسية بالحدثين الذين اعتدنا عليهما و هما **delete-event** و نضع اسم **delete_event** لل **Handler** و الحدث **destroy** و نضع اسم **destroy** لل **Handler**.

بعدها نضيف الحدث **clicked** للزر الذي اضفناه، و ليكن اسم **Handler** هذا الزر هو **clickEvent** بنفس اسم الداله المكتوبه مسبقا.

أما بالنسبة لمربع النص فنضيف له الحدث **key-release-event** و الذي ستجده ضمن قائمة **GtkWidget**، ليكن اسم ال **Handler** هو

keyEvent. هذا كل ما نحتاجه من Glade، خزّن الملف و ليكن اسمه gui.glade

ننتقل الآن إلى النص البرمجي، أسفل الداله clickEvent مباشرة نبدأ بإنشاء كائن الـ Builder ثم نستدعي ملف الواجهه .glade كما تعلمنا مسبقاً، كالتالي :

```
builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );
```

لو لاحظنا نصنا البرمجي سنلاحظ أننا بحاجة إلى ادايتين فقط هما اللتان سنتحكم بهما، الاداة الاولى هي مربع النص، و الثانيه هي النص (Label)، و بالتالي سنستخدم الداله get_object مع هاتين الادائين فقط، كالتالي :

```
window = builder.get_object( 'window1' );
text_entry = builder.get_object( 'entry1' );
status_label = builder.get_object(
'label1' );
```

لا تستغرب، فنحن نقوم بأخذ النافذه دائماً من اجل استدعاء الداله (-: show

نربط الآن الاشارات كما تعلمنا مسبقاً، كالتالي :

```
signals = { "clickEvent" : clickEvent,
            "keyEvent" : keyEvent,
            "destroy" : destroy,
```

```
        "delete_event" :
delete_event}
builder.connect_signals( signals );
```

نُظهر النافذه الرئيسيّه :

```
window.show();
```

و بالطبع لا ننسى الداله الرئيسيّه التي لم نحذفها من الشيفره السابقه :

```
gtk.main();
```

هذا كل شيء! (-:

الشيفره كامله :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete_event( widget, data ):
    return False;

def destroy( widget, data = None ):
    gtk.main_quit();

def clickEvent( widget, data = None ):
    global text_entry, status_label;

    val = text_entry.get_text();
    val = float( val );

    result = ( val * 1.8 ) + 32;
    result = str( result );

    status_label.set_text( result );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
```

```
        clickEvent( None );

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

window = builder.get_object( 'window1' );
text_entry = builder.get_object( 'entry1'
);
status_label = builder.get_object(
'label1' );

signals = { "clickEvent" : clickEvent,
            "keyEvent" : keyEvent,
            "destroy" : destroy,
            "delete_event" :
delete_event}

builder.connect_signals( signals );

window.show();

gtk.main();
```

نقل الآله الحاسبه إلى Glade

كان نقل برنامج التحويل إلى Glade بسيطاً، أليس كذلك؟ (-:

لننقل الآن الآله الحاسبه التي كتبناها في إحدى الاقسام السابقه، الشيفره المصدريه هي نفسها السابقه، سوف نعدلها من اجل العمل مع Glade :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def doMath( widget, data = None ):
    global result_label, first_number,
    second_number, operation;

    operand1 = first_number.get_text();
    operand2 = second_number.get_text();
    op = operation.get_text();

    if ( not operand1 or not operand2
or not op ):
```

```

        result_label.set_text( " يرجى تعبئة "
المعلومات المطلوبه " );
    else:
        operand1 = int( operand1 );
        operand2 = int( operand2 );

        result = operand1;

        if op == "+":
            result += operand2;
        elif op == "-":
            result -= operand2;
        elif op == "*":
            result *= operand2;
        elif op == "/":
            result /= operand2;
        else:
            result_label.set_text( "
العملية التي قمت بإختيارها غير صحيحة " );
            return False;

        result_label.set_text(
str( result ) );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        doMath( None );

# ... #

```



```

win = gtk.Window();

win.set_title( "آله حاسبه" );
win.connect( 'delete_event', delete );
win.connect( 'destroy', des );

# ... #

main_box = gtk.VBox();
win.add( main_box );

# ... #

box1 = gtk.HBox();

# ... #

first_number = gtk.Entry();
box1.pack_start( first_number );
first_number.show();

# ... #

operation = gtk.Entry();
box1.pack_start( operation );
operation.show();

# ... #

second_number = gtk.Entry();
box1.pack_start( second_number );
second_number.show();

```

```

# ... #

first_number.connect( 'key_release_event'
, keyEvent );
operation.connect( 'key_release_event',
keyEvent );
second_number.connect( 'key_release_event'
, keyEvent );

# ... #

main_box.pack_start( box1 );

box1.show();

# ... #

button = gtk.Button( "أظهر الناتج" );
button.connect( 'clicked', doMath );
main_box.pack_start( button );
button.show();

# ... #

result_label = gtk.Label();
main_box.pack_start( result_label );
result_label.show();

# ... #

main_box.show();

win.show();

```

```
gtk.main();
```

هذه هي الشيفره، و اذا حذفنا شيفرة الواجهه تكون الشيفره كالتالي :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def doMath( widget, data = None ):
    global result_label, first_number,
    second_number, operation;

    operand1 = first_number.get_text();
    operand2 = second_number.get_text();
    op = operation.get_text();

    if ( not operand1 or not operand2
or not op ):
        result_label.set_text( " يرجى تعبئة "
المعلومات المطلوبه " );
    else:
```

```

operand1 = int( operand1 );
operand2 = int( operand2 );

result = operand1;

if op == "+":
    result += operand2;
elif op == "-":
    result -= operand2;
elif op == "*":
    result *= operand2;
elif op == "/":
    result /= operand2;
else:
    result_label.set_text( "
العملية التي قمت بإختيارها غير صحيحة
" );
    return False;

    result_label.set_text(
str( result ) );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        doMath( None );

gtk.main();

```

لنتذكر الآن، كيف كانت واجهة الآلة الحاسبة حتى نصممها مع Glade، كانت تنقسم إلى ثلاث أقسام، القسم الأول يحتوي على ثلاث مربعات

نص، القسم الثاني يحتوي على زر مكتوب عليه " اظهر الناتج " ، و القسم الاخير كان يحتوي على نص لإظهار الناتج فيه.

نبدأ مع Glade، و كما تعودنا نضيف دائماً النافذه، و لأنّ برنامجنا يحتوي على 3 اقسام عموديه فإننا سوف نستخدم الصندوق VBox و نحدد الرقم 3، القسم الاول يحتوي على ثلاث مربعات نص بجانب بعضهن، هذا يعني إننا بحاجة إلى صندوق داخل هذا القسم و هذا الصندوق من نوع HBox تجده في القسم Containers بإسم Horizontal Box، و نختار الرقم 3 لاننا بحاجة إلى ثلاث اقسام افقيه، سوف تجد ان القسم الاول انقسم إلى ثلاث اقسام افقيه عن طريق خطوط ظهرت، نضيف الآن مربع نص في كل قسم افقي جديد، نذهب الآن إلى القسم العمودي الثاني و نضيف زرا جديدا، اخيرا نضيف النص (Label) في آخر قسم.

نغيّر عنوان النافذه اولاً إلى "آله حاسبه" و نربطها بالحدثين delete-event و destroy و لتكن اسماء ال Handler نفس اسماء الدوال، delete و .des

نغيّر النص في اعلى الزر إلى " اظهر الناتج " و نربط الزر بالحدث clicked و نسمي ال Handler بإسم الداله doMath.
نحذف الآن النص الموجود في اداة النص.
نربط مربعات النص الثلاث بالحدث key-release-event و نسمي ال Handler بـ keyEvent

هذا كل شيء!، خزّن العمل الآن و سم الملف بالاسم gui، ننتقل الآن إلى النص البرمجي، ننشئ كائن Builder و نستدعي ملف الواجهة أسفل شيفرة الداله keyEvent كالتالي :

```
builder = gtk.Builder();
```

```
builder.add_from_file( 'gui.glade' );
```

نستخدم الدالة `get_object` مع مربعات النص الثلاثة و نص اظهار النتائج بالاضافه إلى النافذه الرئيسيّه :

```
window = builder.get_object( 'window1' );
first_number = builder.get_object(
'entry1' );
operation = builder.get_object( 'entry2'
);
second_number = builder.get_object(
'entry3' );
result_label = builder.get_object(
'label1' );
```

ننشؤ مصفوفة الإشارات و نربطها باستخدام الداله `connect_signals` :

```
signals = { "doMath" : doMath,
            "keyEvent" : keyEvent,
            "des" : des,
            "delete" : delete}
```

```
builder.connect_signals( signals );
```

اخيراً لا ننسى اضافة السطرين :

```
window.show();
gtk.main();
```

هذا كل شيء! تعمل الآن الحاسبه الآن مع Glade، الامور بسيطه
أليست كذلك؟ :-)

الشيفره كامله :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;

# ... #

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def doMath( widget, data = None ):
    global result_label, first_number,
    second_number, operation;

    operand1 = first_number.get_text();
    operand2 = second_number.get_text();
    op = operation.get_text();

    if ( not operand1 or not operand2
or not op ):
        result_label.set_text( " يرجى تعبئة "
```

```

المعلومات المطلوبه" );
    else:
        operand1 = int( operand1 );
        operand2 = int( operand2 );

        result = operand1;

        if op == "+":
            result += operand2;
        elif op == "-":
            result -= operand2;
        elif op == "*":
            result *= operand2;
        elif op == "/":
            result /= operand2;
        else:
            result_label.set_text( "
العملية التي قمت بإختيارها غير صحيحة" );
            return False;

        result_label.set_text(
str( result ) );

def keyEvent( widget, data ):
    key =
gtk.gdk.keyval_name( data.keyval );

    if ( key == 'Return' ):
        doMath( None );

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

```



```

window = builder.get_object( 'window1' );
first_number = builder.get_object(
    'entry1' );
operation = builder.get_object( 'entry2'
);
second_number = builder.get_object(
    'entry3' );
result_label = builder.get_object(
    'label1' );

signals = {    "doMath" : doMath,
               "keyEvent" : keyEvent,
               "des" : des,
               "delete" : delete}

builder.connect_signals( signals );

window.show();

gtk.main();

```

لاحظ مدى بساطة شيفرة Glade مقارنة بالشفيره السابقه.

الفصل الثاني

مدخل إلى SQLite

مقدمه

هناك الكثير من البرامج التي تقوم بإدارة البيانات، حيث تأخذ مدخلات من المستخدم و تخزنها من اجل الاستخدام فيما بعد، و تتيح هذه البرامج للمستخدم التعديل على البيانات المُخزنه او حتى التخلّص منها، مثلا البرامج التي تستخدم في الشركات و التي تدير الشركة من خلالها بيانات الموظفين، كذلك برامج القواميس التي تعطي معاني الكلمات، تستخدم هذه البرامج مُحركات قواعد بيانات لتخزين البيانات و التعامل معها، تقدّم لغة Python العديد من مُحركات قواعد البيانات، و لكننا سوف نستخدم المُحرك SQLite نظرا لبساطته و سهولته، يستخدم هذا المُحرك لغة SQL من اجل التعامل مع البيانات، أفترض هنا ان لديك إلمام بلغة SQL، اذا لم تكن تعرف هذه اللغة انصحك بمراجعة الكتب المُتخصصه او مواقع الانترنت فهي مليئه بدروس SQL التي تشرح التفصيل، استخدم اي محرك بحث و سوف تجد العديد من المواقع على الشبكة.

بالطبع قبل البدء لابد من تثبيت مكتبة SQLite على حاسوبك، سوف تحتاج إلى المكتبة PysQLite فراجع موقعها للحصول عليها و على كيفية تثبيتها.

الدوال الاساسيه

تقدّم المكتبة PySQLite مجموعه من الدوال، و لكننا لن نحتاجها جميعها، سوف نستخدم بعض الدوال الاساسيه التي يحتاجها اي برنامج يعتمد على PySQLite. يمكنك كالعاده القراءه فيما بعد اكثر حول الدوال التي تقدمها PySQLite و قد تجد دوال تهتمك.

نبدأ بالدالتين الأهم و هما `connect` و `cursor`.

الداله الاولى و هي `connect` و تُستخدم للاتصال بقاعدة البيانات، قاعدة البيانات عباره عن ملف يمكنك تخزينه في اي مكان تشاء، و تمرر مسار هذا الملف إلى الداله `connect`. اذا لم تجد الداله `connect` ملف قاعدة البيانات في المسار المطلوب ستقوم بإنشاءه.

الداله الثانيه هي `cursor` و هي داله اساسيه جداً، حيث تأخذ مؤشر SQLite و تعيده إلى متغير، سوف نستخدم هذا المتغير فيما بعد في جميع العمليات الاخرى، مثل عملية إنشاء الاستعلامات او اخذ ناتج عملية الاستعلامات، و هذا مثال لاستخدام هاتين الدالتين :

```
from pysqlite2 import dbapi2 as sqlite;  
  
connect = sqlite.connect( "DB" );  
  
cur = connect.cursor();
```

اولاً نستدعي مكتبة SQLite في السطر الاول، في السطر الثاني نتصل بقاعدة البيانات التي اسمها DB، بما اننا لم نمرر مسار معين و قمنا بوضع

اسم قاعدة البيانات فحسب هذا يعني ان قاعدة البيانات DB مُخزّنه في نفس المجلد الذي يحتوي على ملف بايثون، السطر الاخير يأخذ المؤشر و يخزّنه في المتغير cur. بعدها سنستخدم المتغير cur دائما من اجل العمليات الاخرى.

نعرّج على الدالتين execute و fetchall، بالنسبه للداله الاولى وهي execute نستخدمها لتنفيذ الاستعلامات بلغة SQL حيث نمرر نص الاستعلام كبارامتر للداله. سنستخدم هذه الداله فيما بعد من اجل إنشاء الجدوال و اخذ البيانات و التعديل عليها.

الداله الثانيه وهي fetchall نستخدمها دائما بعد عمل استعلام من نوع SELECT. بحيث نحصل على مصفوفه تحتوي على النتائج.

اخيرا الداله commit و نستخدمها اذا قمنا بأي عملية تغيير في قاعدة البيانات، مثلا اصفنا بيانات او حدّثنا بيانات او انشأنا جدولا جديدا، اذا أجرينا اي تغيير في قاعدة البيانات و لم نستخدم هذه الداله ستضيع جميع التعديلات بمجرد إغلاق البرنامج.

سوف نرى امثله واقعيه لجميع هذه الدوال عندما نكتب مشروعنا في هذا الكتاب إن شاء الله :-).

المثال الاول : برنامج لتخزين الاسماء و عرضها

اصبح لدينا الآن إلمام بكتابة برامج ذات واجهه رسوميه باستخدام GTK،
و لدينا إلمام مسبق بلغة SQL، و تعرفنا على الدوال الاساسيه التي تخص
مكتبة SQLite، اذا نحن الآن جاهزون لكتابة برنامج رسومي يعتمد على
قواعد البيانات لتخزين معلوماته.

سوف نبدأ في مثالنا الاول هذا و سوف يكون بسيطاً جداً، وظيفته هي
تخزين مجموعه من الاسماء و عرضها في قائمه، هذا كل ما هنالك.

نبدأ اولاً بتخطيط جداول قواعد البيانات، نحتاج إلى جدول واحد فحسب،
نخزن في هذا الجدول الاسماء و اسم هذا الجدول هو names، اما
بالنسبه للحقول نحتاج إلى حقلين، الحقل الاول هو id و الذي يخزن رقم
تعريفي مميز، اما الحقل الثاني هو name وهو الاسم الذي نود تخزينه.

الآن لتحدث عن واجهه البرنامج، لاننا نريد البرنامج بسيطاً لن يكون هناك
عدّة نوافذ، اعني اننا لن نستخدم نافذة منفصلة من اجل إضافة اسم جديد،
و لكن سنستخدم النافذه الرئيسييه و نضع فيها قائمة الاسماء و في اسفلها
نضع الادوات اللازمه لإضافة اسم جديد إلى قواعد البيانات.

هذا يعني اننا سوف نقسم واجهه البرنامج إلى خمس اقسام باستخدام
صندوق من نوع VBox، القسم الاول يحتوي على قائمه من نوع
TreeView، القسم الثاني يحتوي على نص (Label) و غيره إلى
"إضافة كلمة جديده"، القسم الثالث يحتوي على مربع نص يتم كتابة
الاسم فيه، اما القسم الرابع فيحتوي على زر مكتوب عليه "موافق"، و

اخيراً القسم الخامس يحتوي على نص فارغ يعرض الحالة، مثلاً اذا نجحت الاضافه يتم عرض ذلك في هذا النص، يُفترض أن تكون واجهة البرنامج كالتالي :



صممها ثم قم بتخزين الملف بإسم gui، لا

نسى الآن ربط النافذه بالدالتين الاساسيتين، للحدث `destroy` نستخدم داله بإسم `delete-event` نستخدم داله بإسم `delete`، بعدها نربط الزر بالحدث `clicked` بالداله `addNewName`، نخزّن الواجهه الآن ثم نعرّج على الملف البرمجي، نكتب الاساسيات التي اعتدنا دائماً على كتابتها :

```
# -*- coding: utf-8 -*-
```

```
import pygtk;
pygtk.require( '2.0' );
import gtk;
import gtk.glade;
from pysqlite2 import dbapi2 as sql;

def delete( widget, data ):
```

```

return False;

def des( widget, data = None ):
    gtk.main_quit();

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

```

مناذاة المكتبات و كتابة الدالتين `des` و `delete` ثم إنشاء كائن `Builder` و إستدعاء ملف الواجهه، نستدعي الآن الادوات التي سنستخدمها داخل النص البرمجي، سنحتاج إلى التعامل مع القائمه و مع مربع النص و مع النص الثاني الذي يبين الحاله، و بالتالي نستدعي هذه الادوات الثلاثة.

```

window = builder.get_object( 'window1' );
treeview = builder.get_object( 'treeview1' );
entry = builder.get_object( 'entry1' );
status = builder.get_object( 'label2' );

```

نربط الاحداث :

```

signals = { "addNewName" : addNewName,
            "des" : des,
            "delete" : delete}

builder.connect_signals( signals );

```

قبل كتابة الداله `addNewName` لا بد من عمل هام جداً، وهو إنشاء قاعدة البيانات و الجدول الذي سوف نستخدمه، من الأفضل إنشاء ملف برمجي جديد يحتوي على شيفرة إنشاء الجدول، نسمي هذا الملف باسم

database_create.py و يكون محتواه كالتالي، نستدعي اولاً مكتبة
: SQLite

```
# -*- coding: utf-8 -*-
```

```
from sqlite3 import dbapi2 as sql;
```

ثم نتصل بقاعدة بيانات إسمها sqlite_db باستخدام الدالة connect
المشروحه سلفاً :

```
con = sql.connect( 'sqlite_db' );
```

نخزن المؤشر في المتغير cur :

```
cur = con.cursor();
```

ثم نمرر جملة SQL الخاصه بإنشاء جدولنا المطلوب إلى الدالة
execute و نستخدم print قبلها حتى نتعرف على حالة الإنشاء :

```
print cur.execute( 'CREATE TABLE names  
(id int,name varchar(255))' );
```

هذا كل شيء، سوف يكون الملف كالتالي :

```
# -*- coding: utf-8 -*-
```

```
from sqlite3 import dbapi2 as sql;
```

```
con = sql.connect( 'sqlite_db' );
```

```
cur = con.cursor();
print cur.execute( 'CREATE TABLE names
(id int,name varchar(255))' );
```

نخزن الملف باسم `database_create.py` ونقوم بتشغيله، بعد تشغيل الملف بنجاح نلاحظ أنّ هناك ملف جديد اسمه `sqlite_db` موجود في مجلد برنامجنا، هذه هي قاعدة البيانات التي تحتوي على الجدول الذي سوف نتعامل معه فيما بعد.

نعود الآن إلى الملف الاساسي للبرنامج وهو `main.py`. قبل كتابة الدالة `addNewName` يجب أن نتصل بقاعدة البيانات كما فعلنا تماما في الملف `database_create.py`، نذهب إلى الملف `main.py` و قبل السطر:

```
builder = gtk.Builder();
```

وهو السطر الخاص بإنشاء كائن `Builder`، نضيف السطرين الخاصين بالاتصال في قاعدة البيانات و اخذ مكان المؤشر:

```
con = sql.connect( 'sqlite_db' );
cur = con.cursor();
```

لاحظ ان اسم قاعدة البيانات هو نفسه `sqlite_db`، نعود الآن إلى الدالة `addNewName`، تقوم هذه الدالة بأخذ المحتوى الموجود داخل مربع النص، و التحقق اذا كان المحتوى فارغا (أي ان مستخدم البرنامج لم يكتب شئاً داخل مربع النص و ضغط على الزر) في هذه الحالة تعرض له رساله في الاسفل تخبره بأنه عليه تعبئة المعلومات، اما اذا كان

الاسم مكتوباً سوف نقوم بإضافة الاسم إلى قاعدة البيانات، نكتب دالتنا في اسفل الداله `des`، نبدأ كالتالي :

```
def addNewName( widget, data = None ):
```

في داخل الداله ستتعامل مع مربع النص الذي سنأخذ منه الاسم المطلوب تخزينه، و سنتعامل مع النص الذي نعرض الحاله من خلاله، كذلك نستخدم قواعد البيانات داخل الداله اذا نحن بحاجة إلى المتغيرين الخاصين بقواعد البيانات و هما `con` و `cur`. و بالتالي سوف نستخدم `global` كما تعودنا مع المتغيرات الاربعه `entry` و `status` و `con` و `cur` كالتالي :

```
global entry, status, con, cur;
```

نستخدم الآن الداله `get_text` من اجل اخذ المحتوى الموجود في صندوق النص و نخزّن قيمه في متغير نسميه `name` :

```
name = entry.get_text();
```

نتحقق الآن. اذا كانت القيمه فارغه نطبع رساله للمستخدم :

```
if ( not name ):  
    status.set_text( 'يرجى كتابة الاسم المطلوب' );
```

اذا لم تكن فارغه نقوم بإضافة الاسم في قاعدة البيانات، اولاً نستخدم الداله `execute` التي تأخذ ممتاً جملة `SQL` الخاصه بإضافة البيانات إلى قاعدة البيانات. تكون دائماً داله `execute` ضمن المتغير الذي يخزّن المؤشر و في حالتنا هذه المتغير هو `cur`. اذا نكتب اولاً السطر الخاص

بإضافة البيانات :

```
insert = cur.execute( 'INSERT INTO  
names(id,name) VALUES(NULL, "' + name +  
"')' );
```

عندما نقوم بأي تعديل على البيانات، سواء أضفنا أو حذفنا أو حدّثنا البيانات عن طريق الدالة `execute`، يجب علينا استدعاء الدالة `commit` حتى نسجّل هذه التغييرات بشكل فعلي في قاعدة البيانات، فإذا قمنا مثلاً بإضافة قيمة جديدة في قاعدة البيانات و لم نقم بإستدعاء `commit` بعدها في حال إغلاق البرنامج تذهب هذه المعلومات التي أضفناها ولا تسجّل في قاعدة البيانات، و بالتالي انتبه دائماً إلى هذه النقطة حتى لا تضيع المعلومات، تقع الدالة دائماً تحت متغير الاتصال و اسم هذا المتغير في حالتنا هو `con` :

```
check = con.commit();
```

الدالة `commit` تُرجع `None` في حال نجاحها، و بالتالي نتحقق من نجاحها و في حال نجاحها نخبر المستخدم بذلك كالتالي :

```
if check == None:  
    status.set_text( 'تم اضافة الاسم بنجاح' );
```

و من اجل التسهيل على المستخدم حتى يتمكن من اضافة اسماء متعدده اسم تلو الآخر نقوم بتفريغ محتوى مربع النص بعد نجاح العمليه كالتالي :

```
entry.set_text( '' );
```

إنتهينا الآن من الدالة addNewName و شيفرتها كالتالي :

```
def addNewName( widget, data = None ):
    global entry, status, con, cur;

    name = entry.get_text();

    if ( not name ):
        status.set_text( ' يرجى كتابة الاسم
المطلوب ' );
    else:
        insert = cur.execute( 'INSERT
INTO names(id,name) VALUES(NULL,'" + name
+ '"' ) );

        check = con.commit();

        if check == None:
            status.set_text( ' تم اضافة الاسم
بنجاح ' );
            entry.set_text( '' );
```

انهيينا الآن جزء هام في البرنامج، تبقى الجزء الثاني وهو عرض قائمه الاسماء، كما تعلم قمنا بإضافة الاداة TreeView من اجل عرض قائمة الاسماء، في الحقيقة تستخدم هذه الاداة في عملها اسلوب يُسمّى بـ Model-View-Controller و نطلق عليه MVC اختصارا، و حتى تتمكن من التعامل بشكل صحيح مع هذه الاداة لابد ان نفهم هذا الاسلوب، الجدير بالذكر إنّ اسلوب MVC يُعتبر أحد أنماط التصميم)

Design Pattern و هي مجموعة من الحلول لمشاكل عامة تواجه المبرمج، في الآونة الأخير كثر استخدام MVC في تطوير المواقع خصوصاً.

يعتمد هذا الاسلوب على تقسيم العمل إلى ثلاث اجزاء.

الجزء الأول هو **Model** و يختص هذا الجزء بالبيانات بمعنى ان البيانات بأنواعها تخزن داخل هذا الجزء.
أما الجزء الثاني وهو **View** فتقع عليه مسؤولية عرض هذه البيانات المُخزنه في **Model**.
أما الجزء الثالث وهو **Controller** فهو مسؤول عن معالجة الاحداث.

الجزءان الاول و الثاني هما الاعم بالنسبه لنا الآن، اذاً يمكننا الاستنتاج انّ الهدف الاساسي من اسلوب MVC هو فصل البيانات عن الطريقة التي تعرض بها.

نعود للإداة **TreeView**، كما ذكرنا ان هذه الاداة تستخدم الاسلوب MVC، تقدّم مكتبة **GTK** فئة اسمها **TreeViewColumn** هذه الفئة هي المسؤولة عن جزء العرض **View**، كما انها تقدّم فئتين **ListStore** و **TreeStore** ويمكن استخدام احدي هاتين الفئتين ك **Model** لتخزين البيانات التي ستعرض في القائمه، تستخدم الفئة الاولى من اجل تخزين البيانات التي ستعرض على شكل قائمه بسيطه، اما الفئة الثانيه فتستخدم من اجل تخزين البيانات التي ستعرض على شكل شجره او بالاحرى التي ستعرض على شكل قائمه تحتوي على هرميات و بيانات تقع تحت بيانات اخرى، و بالطبع لاننا سوف نطبع الاسماء على شكل قائمه بسيطه يجب علينا استخدام الفئة **ListStore**.

نبدأ أولاً بكتابة جزء العرض (View) مع الاداة `TreeViewColumn`، كما ذكرنا ان هناك فئة بنفس الاسم و بالتالي من اجل البدء بجزء العرض نقوم بإنشاء كائن من هذه الاداة و نكلّف متغير ليكن اسمه `col` بها كالتالي :

```
col = gtk.TreeViewColumn( 'الاسم',
gtk.CellRendererText(), text = 0 );
```

نلاحظ إنّ البارامتر يحتوي على عنوان العمود الذي يكون في الاعلى، في كل مره ننشئ كائن جديد من الصنف `TreeViewColumn` فإننا بهذه الطريقة نقوم بإنشاء عمود جديد ضمن قائمتنا، في حالتنا هذه نحن بحاجة إلى عمود واحد فقط لأننا نريد عرض الاسماء فقط، لو كئنا نريد عرض المزيد من المعلومات مثلا الاسم و رقم الهاتف و البريد الالكتروني، سنحتاج إنشاء ثلاث كائنات لتنشئ بدورها ثلاث اعمده.

نعود للسطر السابق، البارامتر الثاني غالبا ما يكون ثابتا، اما البارامتر الثالث تتضح فائدته عندما تكون لدينا عدّة اعمده حيث يأخذ العمود الاول الرقم 0 و العمود الثاني الرقم 1 و هكذا، لن نفصل حاليا هذا الموضوع لاننا سوف نفصله فيما بعد إن شاء الله، في الوقت الحالي لان لدينا عمود واحد و لان هذا العمود هو الاول و بالتالي سوف يأخذ الرقم 0.

بعدها انشأنا العمود وهو الجزء المُسمى بـ `View` بإسلوب MVC الذي تستخدمه الاداة `TreeView` يجب علينا الآن إخبار الاداة `TreeView` عن الكائن الذي يحتوي على الجزء `View` و هو `col` في حالتنا، لمزيد من التوضيح لنفرض ان واجهة برنامجنا تحتوي على قائمتين، هذا يعني أنّ لدينا كائنان من الفئة `TreeView`، و هذان الكائنان بحاجة إلى الجزء `View` الذي ننشئه من خلال الفئة `TreeViewColumn`، لنفرض ان القائمه الاولى تحتوي على عمود واحد اسمه "البريد الالكتروني" كذلك

بالنسبة للقائمه الثانيه لنفرض انها تحتوي على عمود واحد وهو " رقم الهاتف"، ببساطه يمكننا إنشاء كائنين من الصنف `TreeViewColumn` كالتالي :

```
email_col = gtk.TreeViewColumn( ' البريد  
'الالكتروني', gtk.CellRendererText(), text=0 );  
phone_col = gtk.TreeViewColumn( 'رقم الهاتف',  
gtk.CellRendererText(), text=0 );
```

و قبلها كما تعلم لابد ان يكون لدينا كائنين من الصنف `TreeView`. لنفرض ان الاول اسمه `first_tree` و الثاني اسمه `second_tree`. و لكن كيف يمكن للبرنامج ان يعرف ان الكائن `email_col` يخص القائمه الاولى و لا يخص القائمه الثانيه؟ حسنا ماذا لو قررنا جعل القائمه الاولى هي التي تحتوي على عمود " رقم الهاتف" و الثانيه تحتوي على عمود "البريد الالكتروني".

ببساطه الصنف `TreeView` يحتوي على الداله `append_column` هذه الداله تحدد الجزء `View` يخص اي كائن `TreeView`. تستقبل هذه الداله بارامتر وهو اسم الكائن الخاص بالجزء `View`. و بالتالي حتى نضع هذين العمودين كل في قائمته نستخدم هذه الداله كالتالي :

```
first_tree.append_column( email_col );  
second_tree.append_column( phone_col );
```

نعود لبرنامجنا الاساسي، من الشرح السابق نستنتج اننا يجب ان نكلف الكائن `col` بالقائمه التي يُمثلها الكائن `tree` كالتالي :

```
tree.append_column( col );
```


هكذا انتهينا من الجزء **View**، تبقى جزء **Model**.

كما اسلفنا انّ الجزء **Model** هو الخاص بتخزين البيانات، تُقدّم مكتبة **PyGTK** صنفين احدهما خاص بالقوائم البسيطة العادية التي استخدمناها في هذا البرنامج (**ListStore**) و الآخر خاص بالقوائم المركبة و المُعقده و التي يُطلق عليها اسم الاشجار (**TreeStore**). بالطبع سوف نستخدم الصنف **ListStore** حتى ننشئ الجزء **Model** كالتالي :

```
store = gtk.ListStore( str );
```

لنشرح الآن نظام البارامترات هنا، نلاحظ في برنامجنا هذا انّ الجزء **View** يحتوي على عمود واحد فقط وهو عمود الاسم، و البيانات التي ستقع تحت هذا العمود حتما ستكون نصوص (او **string** كما نسميها بالبرمجه). لاحظ الآن البارامتر الذي مرناه عندما نادينا الصنف **ListStore**، قمنا بتمرير بارامتر واحد لان لدينا عمود واحد و لان هذا العمود سوف تقع تحته بيانات من نوع النص فهذا البارامتر يوضح نوع هذه المعلومات عن طريق إرسال اسم النوع، حسنا ماذا لو اضعنا عمودنا الثاني و المعلومات التي تقع هي عمر صاحب الاسم، في هذه الحالة سوف نحتاج لتمرير بارامترين الاول يخص العمود الاول الخاص بالاسماء و بالتالي سوف نرسل **str** اما الثاني فيخصص العمود الثاني الخاص بالاعمار و بالتالي سوف نرسل **int** (رقم صحيح) و هكذا في كل مره نستخدم فيها الصنف **TreeView** لعرض قوائمنا.

هل تتذكر ما فعلناه مع الكائن الذي يمثله المتغير **col**؟ حيث قمنا بتكليف الكائن **col** إلى القائمة التي يمثله المتغير **tree** عن طريق السطر :

```
tree.append_column( col );
```

لنفس السبب الذي شرحناه يجب علينا كذلك تكليف الكائن store إلى القائمة الرئيسية و ذلك عن طريق الداله set_model كالتالي :

```
tree.set_model( store );
```

إتھينا الآن من إعداد الجزء Model، نريد الآن تعبئة البيانات التي سوف تعرض في القائمة، عندما يفتح المُستخدم البرنامج لابد ان تظهر قائمة الاسماء المُخزنه في قواعد البيانات و يمكنه بعدها اضافة المزيد إن اراد ذلك.

كما تعلم لأخذ قائمة الاسماء في قاعدة البيانات سوف نحتاج إلى استخدام مكتبة PysQLite، نستخدم اولا الداله execute لإرسال الاستعلام، نحتاج إلى اخذ جميع البيانات المُخزنه في جدول names و بالتالي لابد من صياغة إستعلام لتحقيق هذا المطلوب بإستخدام لغة SQL :

```
get_names_query = cur.execute( "SELECT *  
FROM names" );
```

نستقبل الآن ناتج الاستعلام عن طريق الداله fetchall و نخزنها في names، الناتج عبارته عن مصفوفة :

```
names = cur.fetchall();
```

اذا كانت هناك معلومات فعلاً فإن حجم names يجب ان يكون اكبر

من 0، و بالتالي لابد ان نتحقق اذا كان هناك معلومات فعلاً ام قاعدة البيانات فارغه كالتالي :

```
if len( names ) > 0:
```

بعدها تأكدنا ان هناك معلومات فعلاً نقوم بقراءة المصفوفه names ووضع محتواها داخل الجزء Model، هذه المصفوفه ثنائية البعد، هناك عدة مدخلات و التي تمثل مدخلات قواعد البيانات وهو عدد متغير يعود إلى البيانات التي اضافها المستخدم، و هناك حقان وهما id و name الموجودان في الجدول names، ما نريده هو قراءة هذه المدخلات كامله و طباعة محتوى الحقل name منها و يكون ذلك كالتالي :

```
k = 0;
while k < len( names ):
    store.append( [ names[ k ][ 1 ] ] );

    k += 1;
```

كما تلاحظ أنشأنا حلقة تعتمد على حجم المصفوفه names و تقرأ البيانات، و استخدمنا الداله append التي يقدمها الصنف ListSore لإضافة البيانات في داخله.

انتهينا الآن بشكل كامل من الجزء Model و الشيفره كامله لهذا الجزء :

```
store = gtk.ListStore( str );

tree.set_model( store );

# ... #
```

```

get_names_query = cur.execute( "SELECT *
FROM names" );
names = cur.fetchall();

if len( names ) > 0:
    k = 0;
    while k < len( names ):
        store.append( [ names[ k ][ 1 ]
] );

        k += 1;

```

هكذا نكون قد انتهينا من البرنامج و لكن هناك مشكله واحده، لا يتوقف برنامجنا اذا لم يتم حلها و لكن يتحسن حلها، جرب اضافه اسم جديد من خلال البرنامج، لاحظ انه على الرغم من النجاح في تخزين الاسم بقاعدة البيانات و لكنه لا يظهر في القائمه مباشره بعد إضافته بنجاح، و لكن يجب علينا إغلاق البرنامج ثم تشغيله مره اخرى حتى يظهر الاسم الجديد ضمن القائمه.

الحل كما خمنت بسيط جداً، نتوجه إلى الداله `addNewName` الخاصه بإضافة الاسماء إلى قاعدة البيانات، نضيف المتغير `store` ضمن قائمه المتغيرات التي تقع بعد `global` حتى يصبح السطر كالتالي :

```

global entry, status, con, cur, store;

```

نتوجه الآن إلى الشرط الذي يتحقق من نجاح اضافه الاسم وهو :

```

if check == None:
    status.set_text( ' تم اضافه الاسم بنجاح ' );

```

```
entry.set_text( '' );
```

نستخدم في نهايته الدالة `append` التي تحدثنا عنها منذ قليل لإضافة الاسم إلى القائمة كالتالي :

```
store.append( [ name ] );
```

ستصبح الشيفره بهذا الشكل :

```
if check == None:  
    status.set_text( 'تم اضافة الاسم بنجاح' );  
    entry.set_text( '' );  
    store.append( [ name ] );
```

جرب الآن و لاحظ ان الاسم يُضاف مباشرة إلى القائمة بدون الحاجه إلى إغلاق البرنامج و تشغيله، حسنا و اخيرا انتهينا من هذا المثال
(الحمد لله :-)

الشيفره الكامله للبرنامج :

```
# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;
import gtk.glade;
from pysqlite2 import dbapi2 as sql;

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def addNewName( widget, data = None ):
    global entry, status, con, cur,
store;

    name = entry.get_text();

    if ( not name ):
        status.set_text( 'يرجى كتابة الاسم المطلوب'
);
    else:
        insert = cur.execute( 'INSERT
INTO names(id,name) VALUES(NULL,"' + name
+ '")' );

        check = con.commit();
```

```

        if check == None:
            status.set_text( ' تم اضافة الاسم
بنجاح ' );
            entry.set_text( '' );
            store.append( [ name ] );

con = sql.connect( 'sqlite_db' );
cur = con.cursor();

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

window = builder.get_object( 'window1' );
tree = builder.get_object( 'treeview1' );
entry = builder.get_object( 'entry1' );
status = builder.get_object( 'label2' );

signals = { "addNewName" : addNewName,
            "des" : des,
            "delete" : delete}

builder.connect_signals( signals );

# ... #

col = gtk.TreeViewColumn( 'الاسم',
gtk.CellRendererText(), text = 0 );

tree.append_column( col );

# ... #

```

```
store = gtk.ListStore( str );
tree.set_model( store );

# ... #

get_names_query = cur.execute( "SELECT *
FROM names" );
names = cur.fetchall();

if len( names ) > 0:
    k = 0;
    while k < len( names ):
        store.append( [ names[ k ][ 1 ]
] );
        k += 1;

# ... #

window.show();

gtk.main();
```


الفصل الثالث

إنشاء مشروع

برنامج ادارة بيانات موظفين في شركة ما

لنبدأ الآن ببناء مشروع كامل نستخدم فيه ما تعلمناه. لاحظ أنّ هذا الفصل يعتمد بشكل كبير على شرح مثال برنامج الأسماء الموجود في الفصل الثاني و بالتالي تأكد من فهمك الجيد له.

المشروع ببساطه برنامج لإدارة بيانات موظفين، يشابه إلى حد كبير مثال برنامج الأسماء الذي كتبناه، و لكن هناك المزيد من البيانات للتعامل معها و هي اسم الموظف، عمره، راتبه، و سوف يكون للبرنامج اكثر من نافذه بعكس المثال السابق الذي كان يحتوي على نافذه واحده تعرض البيانات و تتم الاضافه من خلالها.

في هذا المشروع تحتوي النافذه الرئيسيه على قائمه بأسماء الموظفين و اعمارهم و رواتبهم، في اسفل النافذه ثلاث ازرار الاول " اضافة موظف " و يفتح نافذه جديده تطلب بيانات الموظف الجديد، اما الزر الثاني " تحديث البيانات " بحيث يختار المستخدم الموظف المطلوب من القائمه ثم يضغط على هذا الزر لتحريير بيانات الموظف، اما الزر الاخير " حذف الموظف " الذي يعمل بنفس طريقة الزر السابق و لكنه يحذف الموظف بدلا من تحريير معلوماته.

اربط الحداث الاساسيين و هما delete-event مع الداله delete و destroy مع الداله des، و الآن اربط احداث الازرار، اربط الحدث clicked لزر اضافة الموظفين مع داله يا اسم addNewEmp اما زر التحديث فأربط الحدث clicked مع داله يا اسم updateEmp، اما الزر الاخير فأربط الحدث clicked مع داله يا اسم deleteEmp.

خزّن الملف في مجلد البرنامج و ليكن اسمه `gui.glade`

نشرع الآن بالشفيره. نبدأ أولاً مع الملف `database_create.py` و الذي يُنشئ جدول قاعدة البيانات كما مر عليك عندما انشأنا المثال الاول :

```
# -*- coding: utf-8 -*-  
  
from pysqlite2 import dbapi2 as sql;  
  
con = sql.connect( 'employees' );  
cur = con.cursor();  
  
print cur.execute( 'CREATE TABLE names  
(id int,name varchar(255),salary int,age  
int)' );
```

لاحظ اننا انشأنا قاعدة بيانات اسمها `employees` تحتوي على جدول به اربع حقول `id` و `name` و `salary` و `age` و لا تخفى عليك انواعها و الهدف منها، نشغل الآن الملف `database_create.py` حتى يتم إنشاء قاعدة البيانات.

نبدأ بملفنا البرمجي `main.py`، نبدأ بمناداة المكتبات :

```
# -*- coding: utf-8 -*-  
  
import pygtk;  
pygtk.require( '2.0' );  
import gtk;  
import gtk.glade;  
from pysqlite2 import dbapi2 as sql;
```

و دائماً الدالتين delete و des (-:

```
def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();
```

نتصل بقاعدة البيانات :

```
con = sql.connect( 'employees' );
cur = con.cursor();
```

نشؤ كائن Builder و نستدعي ملف Glade :

```
builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );
```

سوف نحتاج للتعامل مع جميع الادوات في داخل الشيفره البرمجيه و بالتالي نحتاج للداله get_object :

```
window = builder.get_object( 'window1' );
tree = builder.get_object( 'treeview1' );
add_emp = builder.get_object( 'button1'
);
update_emp = builder.get_object(
'button2' );
delete_emp = builder.get_object(
'button3' );
```

المتغير window مسؤول عن النافذة الرئيسي، المتغير tree مسؤول عن القائمة، المتغير add_emp مسؤول عن زر "إضافة موظف"، المتغير update_emp مسؤول عن زر "تحديث البيانات"، المتغير delete_emp مسؤول عن زر "حذف موظف".

نربط الاشارات :

```
signals = { "addNewEmp" : addNewEmp,  
            "updateEmp" : updateEmp,  
            "deleteEmp" : deleteEmp,  
            "des" : des,  
            "delete" : delete}
```

```
builder.connect_signals( signals );
```

نضيف في نهاية الملف :

```
window.show();
```

```
gtk.main();
```

قبل السطران السابقان، نبدأ بإنشاء العواميد في قائمتنا كما تعلمنا عندما
انشأنا المثال السابق، العمود الاول خاص بالاسم :

```
col1 = gtk.TreeViewColumn( 'الاسم',  
gtk.CellRendererText(), text = 0 );
```

كما ذكرنا، البارامتر الثالث يحتوي على القيمة 0 بالنسبة للعمود الاول و
كلما زاد عمود نزيد بالرقم، هذا يعني ان العمود الثاني ستكون قيمة

البارامتر الثالث الخاص به هي 1 و الثالث يأخذ القيمة 2 و هكذا، نضيف الآن العمود الثاني :

```
col2 = gtk.TreeViewColumn( 'الراتب',  
gtk.CellRendererText(), text = 1 );
```

ثم العمود الثالث :

```
col3 = gtk.TreeViewColumn( 'العمر',  
gtk.CellRendererText(), text = 2 );
```

نضيف العواميد إلى القائمة :

```
tree.append_column( col1 );  
tree.append_column( col2 );  
tree.append_column( col3 );
```

انتهينا الآن من العواميد او بالاحرى الجزء View، لنبدأ بجزء التخزين Model، بالطبع سوف نستخدم الصنف ListStore لان قائمتنا عاديه و بسيطه :

```
store = gtk.ListStore( str, int, int );
```

اذا قارنًا هذا السطر بالسطر المكتوب بالمثل السابق نلاحظ إننا اضفنا بارامترين و بالتالي اصبحت لدينا ثلاث بارامترات ممرره، اما المثل السابق احتوى فقط على بارامتر واحد، لايد و انك تعرف السبب لاننا شرحناه سابقا :-). السبب هو انه لدينا ثلاث عواميد هنا و بالتالي كل بارامتر يخص عمود من هذه العواميد و يحدد نوعية البيانات التي تقع

تحتة، و السطر السابق يُخبر المكتبة بأن العمود الاول يحتوي على نصوص، العمود الثاني يحتوي على اعداد صحيحة و كذلك بالنسبه للعمود الثالث.

نُحدد الآن الكائن الذي تنتمي إليه هذه المعلومات :

```
tree.setModel( store );
```

نأخذ قائمة الاسماء كما فعلنا في المثال السابق :

```
get_names_query = cur.execute( "SELECT *  
FROM names" );  
names = cur.fetchall();
```

اذا كان هناك بيانات فعلاً اضفها إلى القائمة :

```
if len( names ) > 0:  
    k = 0;  
    while k < len( names ):  
        store.append( [ names[ k ][ 1 ],  
names[ k ][ 2 ], names[ k ][ 3 ] ] );  
  
        k += 1;
```

لاحظ كيف اصبحت المصفوفة المُمرره إلى الداله `append` بوجود عواميد متعدد، كما رأيت التعامل مع عدّة عواميد موضوع بسيط و طبقناه هنا لتتعرف على مدى بساطته (-):

نبدأ الآن بالجزئيه الاخرى وهي كتابة الدوال التي تخص الازرار، لنكتب

أولاً دالة إضافة موظف جديد.

عندما يضغط المستخدم على هذا الزر، نفتح له نافذة جديدة تحتوي على ثلاث مربعات نص الأولى لكتابة اسم الموظف، الثاني لكتابة راتب الموظف، الأخير لكتابة عمر الموظف، و أسفلهم زر مكتوب عليه "موافق" يضغط عليه المستخدم عندما ينتهي من تعبئة المعلومات حتى تتم إضافة المعلومات إلى قاعدة البيانات، و أسفل هذا الزر نص الحالة الذي يكتب فيه "تم إضافة المعلومات بنجاح" عند نجاح إضافة المعلومات، صمم هذه النافذة عن طريق Glade و خزّن الملف في مجلد البرنامج و ليكن اسمه emp.glade

لنسمي مربع النص الخاص بإسم الموظف بـ emp_name اما المربع الخاص بالراتب بـ salary اما المربع الخاص بالعمر بـ age. زر موافق نسميه insert_button و أخيراً نص الحالة نسميه status.

لا بد من لفت نظرك إلى شئ هام جداً الربط بالحدثين delete-event و destroy هنا غير مهم لان نافذتنا التي نصممها الآن نافذة فرعيه و ليست النافذة الرئيسييه، هذا يعني انها نافذة تظهر عندما نضغط زر محدد في النافذة الرئيسييه، اذا حصل و ربطنا الحدثين في النافذة الفرعيه سوف يؤدي إغلاق النافذة الفرعيه إلى إغلاق البرنامج بالكامل و هذا الذي لا يجب أن يحدث، إننا بحاجة إلى الربط بحدث واحد فقط وهو clicked الخاص بزر موافق لربطه بـ Emp.

لنتعلم الآن كيف يمكننا إظهار هذه النافذة و جعلها تؤدي وظيفتها بعد الضغط على زر "إضافة موظف" في النافذة الرئيسييه، نبدأ أولاً بتعريف :

addNewEmp :


```
def addNewEmp( widget, data = None ):
```

ببساطه لإظهار نافذه فرعيه مُخزنه في ملف منفصل نعمل كما فعلنا مع النافذه الرئيسيه تماما، ننشؤ كائن Builder ثم نستدعي ملف Glade و الذي في حالتنا هذه سميناه emp.glade :

```
add_builder = gtk.Builder();  
add_builder.add_from_file(  
'emp.glade' );
```

نكُلف النافذه بمتغير معين ليكن اسمه add_window :

```
add_window = add_builder.get_object(  
'window1' );
```

ثم نطلب إظهار النافذه :

```
add_window.show();
```

هل رأيت كم العمليه بسيطه :-). نكمل الآن اخذ الادوات التي سنحتاج لإستخدامها في الشيفره المصدريه بعد السطر :

```
add_window = add_builder.get_object(  
'window1' );
```

نأخذ كل من مربعات النص، و زر الموافقه، و نص الحاله ولا يخفى عليك السبب :

```
emp_name = add_builder.get_object(
```

```
'emp_name' );  
salary = add_builder.get_object( 'salary'  
);  
age = add_builder.get_object( 'age' );  
status = add_builder.get_object( 'status'  
);
```

نربط الاحداث، و لكن ربط الاحداث في هذه المره يختلف قليلاً، لنرى الشيفره اولا :

```
signals = { "Emp" : ( insertEmp,  
emp_name, salary, age, status ) }
```

```
add_builder.connect_signals( signals );
```

من الاسطر السابقه تستنتج اننا عندما نضغط على زر الموافقه يتم استدعاء داله باسم `insertEmp` و هي المسؤوله عن إضافة بيانات الموظف إلى قاعدة البيانات، و لكن لماذا في هذه المره بالذات و وضعنا الداله داخل مصفوفه مع مجموعه من الكائنات التي عرفناها مسبقا؟

قبل كل شئ لنحدد ما الذي نريد الوصول له من الادوات من داخل الداله `insertEmp`. سوف نحتاج للوصول إلى مربعات النص الثلاث لإستخراج المعلومات منهن و اضافه المعلومات في قاعدة البيانات، بالإضافة إلى ذلك سوف نحتاج إلى نص الحاله لأننا سنغيره في حال نجاح عملية الاضافه او فشلها، جميع هذه الادوات يمكن الوصول لها من خلال الكائنات التاليه : `emp_name` لإسم الموظف، `salary` لراتبه، `age` لعمره، `status` لنص الحاله.

الحل الذي اعتدنا على استخدامه هو استخدام `global`. هذا يعني اننا لو اردنا تطبيق هذا الحل في حالتنا هذه سوف يكون اول سطر في الداله `insertEmp` مشابه للتالي :

```
global emp_name, salary, age, status;
```

و لكن هذا السطر لن يعمل في هذه الحاله! لماذا يا ترى؟ ببساطه هذه الكائنات الاربعه عباره عن متغيرات محليه (Local variables) و ليست متغيرات عامه (Global variables)، المتغيرات المحليه هي المتغيرات التي تم تعريفها في داخل داله معينه و لن يستطيع اي جزء آخر من البرنامج الوصول لها بشكل مباشر ما عدا الداله نفسها، اما المتغيرات

العامه فيتم تعريفها خارج جميع الدوال و بالتالي يمكن لجميع الدوال الوصول لها من خلال وضع اسمها بعد `global`. جرب بنفسك المثال التالي :

```
def A():
    y = 10;
    B();
def B():
    global x, y;

    print x;
    print y;

x = 5;

A();
```

و لاحظ الخطأ الذي يظهر لك، يطبع البرنامج اولاً الرقم 5 بنجاح و لكنه عندما يصل إلى `y` فإنه يُظهر خطأ يقول ان `y` ليست موجوده. هذه هي نفس الحاله التي تمر بنا الآن و الحل بسيط ان شاء الله، يكمن الحل في إرسال هذه الكائنات على شكل بارامترات إلى الداله `InsertEmp` و يتم إرسال بارامترات إلى الداله التي نناديها عند وقوع الحدث مثلما رأيت منذ قليل، مصفوفه نضع فيها اسم الداله اولاً و بعدها البارامترات بالترتيب، يمكننا استخدام الطريقه المشروحه في " الاشارات و الدوال Callback " من هذا الكتاب، عموماً سيكون تعريف الداله

: `InsertEmp`

```
def insertEmp( widget, emp_name, salary,
age, status ):
```

يمكننا القول أنّ الداله `addNewEmp` خاصه بتجهيز و إظهار نافذة الاضافه و الداله `insertEmp` خاصه بإضافة المعلومات التي تزودها بها الداله الاولى، لنكمل، بما اننا نريد الوصول إلى الجزء `Model` في قائمتنا لاننا نريد اضافة الموظف مباشرة إلى القائمه عند النجاح في اضافته إلى قاعدة البيانات وبما ان الجزء `Model` يُمثله الكائن `store` وهو عباره عن متغير عام و ليس محلي سوف نقوم بإستخدام `global` معه حتى نصل إليه :

```
global store;
```

بغرض الاختصار و الأمثله نُخزّن القيم في متغيرات منفصله :

```
name_val = emp_name.get_text();
salary_val = salary.get_text();
age_val = age.get_text();
```

نقوم بعملية اختبار للقيم و إخبار المستخدم في حال فشل الاختبار :

```
if not name_val:
    status.set_text( ' يرجى كتابة الاسم ' );
elif not salary_val:
    status.set_text( ' يرجى كتابة الراتب ' );
elif not age_val:
    status.set_text( ' يرجى كتابة العمر ' );
```

نُكمل السلسله الشرطيه بإضافة `else` و الشيفره الحقيقيه اسفلها، اول سطر يكون الاستعلام الذي يُدخل المعلومات إلى قاعدة البيانات :

```

else:
    insert = cur.execute( 'INSERT INTO
names(id, name, salary, age) VALUES(NULL, "'
+ name_val + '", "' + salary_val + '", "' +
age_val + '" ) );

```

بعدها نتأكد من اضافة البيانات إلى قاعدة البيانات :

```

check = con.commit();

```

اذا تم إدخال البيانات بشكل صحيح يجب ان نقوم بالتالي : اولاً طباعة ان العملية تمت بنجاح في نص الحالة ثم تفريغ المربعات من المعلومات القديمة حتى تكون اضافة معلومات موظف جديد اسهل مثلما فعلنا مسبقاً، و اخيراً اضافة البيانات مباشرة إلى قائمتنا بدلاً من الحاجه إلى إغلاق البرنامج و اعادة فتحه مره اخرى حتى تظهر المعلومات التي اضعناها :

```

if check == None:
    status.set_text( 'تم اضافة الموظف بنجاح' );
    emp_name.set_text( '' );
    salary.set_text( '' );
    age.set_text( '' );
    store.append( [ name_val,
int( salary_val ), int( age_val ) ] );

```

لاحظ اننا استخدمنا الداله int لتحويل البيانات في هذه المتغيرات إلى نوع integer.

انتهينا الآن من كتابة ميزة اضافة الموظف إلى قاعدة البيانات، تبقى تحرير معلومات الموظف و حذفه، عندما يريد المستخدم تحرير معلومات

الموظف يجب عليه اختيار الموظف المطلوب من القائمه ثم الضغط على زر تحديث البيانات، عند الضغط على هذا الزر ستظهر نافذه مشابهه لنافذة اضافة الموظف و لكن مربعات النص فيها تحتوي على معلومات الموظف المُختار، محرر المستخدم بيانات الموظف كما يشاء ثم يضغط على زر الموافقه ليتم تحديث المعلومات في قواعد البيانات، يعمل الحذف بنفس الاسلوب تقريبا، يختار المستخدم الموظف المطلوب ثم يضغط على زر حذف الموظف فيتم حذف الموظف من قواعد البيانات و من القائمه.

لنبدأ اولاً بكتابة شيفرة تحرير معلومات موظف، تقدم مكتبة PyGTK فئة `TreeSelection` و من خلال هذه الفئة يمكننا التعامل مع الصف الذي يختاره المستخدم من القائمه، في حالتنا هنا يختار المستخدم الموظف المطلوب تحديث معلوماته ثم يضغط على زر "تحديث البيانات" سيسبب الضغط على هذا الزر مناداة داله باسم `updateEmp` يجب ان تقوم هذه الداله بالتالي : تتعرف على اسم الموظف الذي تم اختياره حتى تأخذ بياناته من قاعدة البيانات و حتى تستخدم الاسم في جملة SQL الخاصه بعملية التحديث.

حسناً السؤال هنا كيف يمكننا اخذ بيانات الموظف الذي اختاره المستخدم باستخدام الفئة `TreeSelection`، الفئة الرئيسية `TreeView` توفر لنا داله اسمها `get_selection` ترجع هذه الداله كائن من نوع `TreeSelection` و يمكننا من خلال هذا الكائن الوصول إلى بيانات الصف المُختار، لنعرّف اولاً الداله `updateEmp` :

```
def updateEmp( widget, data = None ):
```

نستخدم الداله `get_selection` التي تحدثنا عنها منذ برهه لناخذ الكائن المطلوب و نضعه في متغير نسميه `select` :

```
select = tree.get_selection();
```

الآن، يعتبر المتغير `select` كائنًا من الفئة `TreeSelection`، تُقدّم لنا هذه الفئة داله اسمها `get_selected` و التي ترجع مصفوفه تحتوي على مُدخلين الاول من نوع `gtk.TreeModel` وهو الجزء `Model` من القائمه التي اخترنا منها الموظف و يساوي في حالتنا هذه المتغير `store`، اما المُدخل الثاني من نوع `gtk.TreeIter` و الذي يُمثل الصف الذي اختاره المستخدم من ضمن الصفوف الموجوده في قائمتنا :

```
ar = select.get_selected();
```

سنستخدم الآن الداله التي توفرها لنا الفئة `TreeModel` وهي `get_value` لأخذ اسم الموظف الذي اختاره المستخدم، تأخذ هذه الداله بارامترين الاول هو الصف المطلوب وهو الموجود في `ar[1]` كما ذكرنا، و البارامتر الثاني هو العمود المطلوب، و من خلاله يمكننا تحديد ما هي القيمة التي نريد اخذها بالضبط، اسم الموظف ام عمره ام راتبه، سوف نحتاج إلى اسمه هنا و الاسم موجود في العمود رقم `0` و بالتالي قيمة البارامتر الثاني تساوي `0`، يمكننا هنا إما استخدام الكائن `store` او استخدام `ar[0]` الذي ارجعته الداله `get_selected`. حسنا سوف استخدم `ar[0]` هنا حتى نستفيد منها على الاقل (-):

```
name = ar[ 0 ].get_value( ar[ 1 ], 0 );
```

اصبح اسم الموظف مُخزناً في المتغير `name`، يمكننا استخدام هذا المتغير الآن في جملة `SQL` لأخذ معلومات الموظف من قواعد البيانات :

```
info_query = cur.execute( 'SELECT * FROM
```



```
names WHERE name="" + name + "" );
```

نُحضر جميع المعلومات و نخزنها في المتغير info :

```
info = cur.fetchall();
```

بنفس الطريقة التي فتحنا بها نافذه جديده لإضافة موظف جديد نفتح نافذه جديده لعرض بيانات الموظف المُختار، سوف نستخدم نفس النافذه التي صممناها لنافذة اضافة موظف وهي emp.glade لانه لا فرق بينهما بتاتا :

```
edit_builder = gtk.Builder();  
edit_builder.add_from_file( 'emp.glade' )  
;
```

نأخذ الادوات التي نحتاج ان نتعامل معها في الشيفره البرمجيّه :

```
edit_window = edit_builder.get_object(  
'window1' );  
emp_name = edit_builder.get_object(  
'emp_name' );  
salary = edit_builder.get_object(  
'salary' );  
age = edit_builder.get_object( 'age' );  
status = edit_builder.get_object(  
'status' );
```

هنا نربط الحدث Emp بالداله editEmp و هذه الداله مشابهه للداله insertEmp إلى حد كبير و تستقبل نفس البارامترات تقريبا :

```
signals = { "Emp" : ( editEmp, info[ 0 ][
1 ], emp_name, salary, age, status ) }

edit_builder.connect_signals( signals );
```

لاحظ البارامتر الاول الذي تستقبله الداله editEmp وهو اسم الموظف المُراد تحرير بياناته، البارامتر الثاني هو الكائن الذي يتحكم في مربع النص الخاص بإسم الموظف، البارامتر الثالث هو الكائن الذي يتحكم في مربع النص الخاص بالراتب، البارامتر الرابع هو الكائن الذي يتحكم في مربع النص الخاص بالعمر، و البارامتر الخامس هو الكائن الذي يتحكم في نص الحاله.

يغير المستخدم البيانات كما يشاء ثم يضغط على زر موافق لتحديث المعلومات في قاعدة البيانات و الداله editEmp هي المسؤوله عن عملية تحديث البيانات في قاعدة البيانات.

نعرض الآن معلومات الموظف، كل معلومه في محلها الصحيح :

```
emp_name.set_text( info[ 0 ][ 1 ] );
salary.set_text( str( info[ 0 ][ 2 ] ) );
age.set_text( str( info[ 0 ][ 3 ] ) );
```

و اخيراً نُظهر النافذه :

```
edit_window.show();
```

شيفرة updateEmp كامله :

```

def updateEmp( widget, data = None ):
    select = tree.get_selection();
    ar = select.get_selected();
    name = ar[ 0 ].get_value( ar[ 1 ], 0
);

    info_query = cur.execute( 'SELECT *
FROM names WHERE name="' + name + '"' );
    info = cur.fetchall();

    edit_builder = gtk.Builder();
    edit_builder.add_from_file(
'emp.glade' );

    edit_window =
edit_builder.get_object( 'window1' );
    emp_name = edit_builder.get_object(
'emp_name' );
    salary = edit_builder.get_object(
'salary' );
    age = edit_builder.get_object( 'age'
);
    status = edit_builder.get_object(
'status' );

    signals = { "Emp" : ( editEmp, info[
0 ] [ 1 ], emp_name, salary, age, status )
}

    edit_builder.connect_signals( signals
);

```

```

    emp_name.set_text( info[ 0 ][ 1 ] );
    salary.set_text( str( info[ 0 ][ 2 ]
) );
    age.set_text( str( info[ 0 ][ 3 ] )
);

edit_window.show();

```

نبدأ بكتابة الدالة الحقيقيه لتحديث المعلومات وهي editEmp :

```

def editEmp( widget, old_name, emp_name,
salary, age, status ):

```

نأخذ المعلومات الجديده التي كتبها المستخدم :

```

name_val = emp_name.get_text();
salary_val = salary.get_text();
age_val = age.get_text();

```

نتأكد من انّ المستخدم لم يترك شيئاً خالياً :

```

if not name_val:
    status.set_text( 'يرجى كتابة الاسم' );
elif not salary_val:
    status.set_text( 'يرجى كتابة الراتب' );
elif not age_val:
    status.set_text( 'يرجى كتابة العمر' );

```

نكمل هذه السلسله الشرطيه بـ else و يكون تحتها الاستعلام الذي

يُحدّث البيانات :

```
else:  
    update = cur.execute( 'UPDATE names  
SET name="' + name_val + '", salary="' +  
salary_val + '", age="' + age_val + '"  
WHERE name="' + old_name + '"' );
```

نغيّر المعلومات بشكل فعلي :

```
check = con.commit();
```

نتحقق من نجاح العمليه و نطبع ذلك للمستخدم :

```
if check == None:  
    status.set_text( ' تم تحديث الموظف بنجاح '  
);
```

هذا كل شيء، لاحظ مدى التشابه بين هذه الداله و بين اختها insertEmp. تكرار الشيفرات بالطبع تعتبر من العادات البرمجية السيئة، و لو كان هذا البرنامج حقيقيا و لم يكن مجرد مثال تعليمي فمن الأفضل وضع الشيفره المكرره بين inserEmp و editEmp في داله منفصله ثم استدعاءها داخل هاتين الدالتين، و لكن لأنّ هذا الموضوع خارج عن نطاق موضوع كتابنا و لأنّ المثال تعليمي فإننا لن نقوم بهذه الخطوه هنا.

الشيفره كامله للداله empEmp :

```
def editEmp( widget, old_name, emp_name,
```

```

salary, age, status ):
    name_val = emp_name.get_text();
    salary_val = salary.get_text();
    age_val = age.get_text();

    if not name_val:
        status.set_text( 'يرجى كتابة الاسم' );
    elif not salary_val:
        status.set_text( 'يرجى كتابة الراتب' );
    elif not age_val:
        status.set_text( 'يرجى كتابة العمر' );
    else:
        update = cur.execute( 'UPDATE
names SET name="' + name_val +
'", salary="' + salary_val + '", age="' +
age_val + '" WHERE name="' + old_name +
'"' );

        check = con.commit();

        if check == None:
            status.set_text( 'تم تحديث معلومات
الموظف بنجاح' );

```

الجزء الاخير في هذا المشروع (-) الداله deleteEmp وهي المسؤوله عن حذف الموظف، يختاره المستخدم ثم يضغط على الزر ليحذفه، كما فعلنا تماما مع updateEmp يجب علينا اخذ اسم الموظف الذي نريد حذفه حتى نستخدمه في استعلام الحذف، نعم كما خمنت تماما سوف نستخدم الصنف TreeSelection (-)، اولا التعريف :

```
def deleteEmp( widget, data = None ):
```

نأخذ اسم الموظف بنفس الاسلوب السابق :

```
select = tree.get_selection();  
ar = select.get_selected();  
name = ar[ 0 ].get_value( ar[ 1 ], 0 );
```

نستخدم المتغير name في الاستعلام :

```
delete = cur.execute( 'DELETE FROM names  
WHERE name="' + name + "' );
```

نحدّث قاعدة البيانات :

```
check = con.commit();
```

حسناً الذي نريده الآن بعدما يتم حذف الموظف بنجاح من قاعدة البيانات هو اختفاءه من القائمه كذلك، نستخدم الداله remove التي تقدمها الفئة ListStore لإزالة الموظف من القائمه، تحتاج هذه الداله لبارامتر واحد وهو الصف المطلوب حذفه و كما تعلم هو موجودٌ في ar[1] و بالتالي شيفرتنا ستكون هكذا :

```
if check == None:  
    store.remove( ar[ 1 ] );
```

بفضل الله و توفيقه نكون هكذا انتهينا من هذا المشروع (-)، الشيفره الكامله :

```

# -*- coding: utf-8 -*-

import pygtk;
pygtk.require( '2.0' );
import gtk;
import gtk.glade;
from pysqlite2 import dbapi2 as sql;

def delete( widget, data ):
    return False;

def des( widget, data = None ):
    gtk.main_quit();

def addNewEmp( widget, data = None ):
    add_builder = gtk.Builder();
    add_builder.add_from_file(
'emp.glade' );

    add_window = add_builder.get_object(
'window1' );
    emp_name = add_builder.get_object(
'emp_name' );
    salary = add_builder.get_object(
'salary' );
    age = add_builder.get_object( 'age'
);
    status = add_builder.get_object(
'status' );

    signals = { "Emp" : ( insertEmp,
emp_name, salary, age, status ) }

```



```

    add_builder.connect_signals( signals
);

add_window.show();

def insertEmp( widget, emp_name, salary,
age, status ):
    global store;

    name_val = emp_name.get_text();
    salary_val = salary.get_text();
    age_val = age.get_text();

    if not name_val:
        status.set_text( 'يرجى كتابة الاسم' );
    elif not salary_val:
        status.set_text( 'يرجى كتابة الراتب' );
    elif not age_val:
        status.set_text( 'يرجى كتابة العمر' );
    else:
        insert = cur.execute( 'INSERT
        INTO names(id,name,salary,age)
        VALUES(NULL, "' + name_val + '", "' +
        salary_val + '", "' + age_val + '" )' );

        check = con.commit();

        if check == None:
            status.set_text( 'تم اضافة الموظف
بنجاح' );

            emp_name.set_text( '' );
            salary.set_text( '' );
            age.set_text( '' );

```

```

        store.append( [ name_val,
int( salary_val ), int( age_val ) ] );

def updateEmp( widget, data = None ):
    select = tree.get_selection();
    ar = select.get_selected();
    name = ar[ 0 ].get_value( ar[ 1 ], 0
);

    info_query = cur.execute( 'SELECT *
FROM names WHERE name="' + name + '"' );
    info = cur.fetchall();

    edit_builder = gtk.Builder();
    edit_builder.add_from_file(
'emp.glade' );

    edit_window =
edit_builder.get_object( 'window1' );
    emp_name = edit_builder.get_object(
'emp_name' );
    salary = edit_builder.get_object(
'salary' );
    age = edit_builder.get_object( 'age'
);
    status = edit_builder.get_object(
'status' );

    signals = { "Emp" : ( editEmp, info[
0 ][ 1 ], emp_name, salary, age, status )
}

    edit_builder.connect_signals( signals

```

```

    );

    emp_name.set_text( info[ 0 ][ 1 ] );
    salary.set_text( str( info[ 0 ][ 2 ] )
) );
    age.set_text( str( info[ 0 ][ 3 ] )
);

    edit_window.show();

def editEmp( widget, old_name, emp_name,
salary, age, status ):
    name_val = emp_name.get_text();
    salary_val = salary.get_text();
    age_val = age.get_text();

    if not name_val:
        status.set_text( 'يرجى كتابة الاسم' );
    elif not salary_val:
        status.set_text( 'يرجى كتابة الراتب' );
    elif not age_val:
        status.set_text( 'يرجى كتابة العمر' );
    else:
        update = cur.execute( 'UPDATE
names SET name="' + name_val +
'", salary="' + salary_val + '", age="' +
age_val + '" WHERE name="' + old_name +
'"' );

        check = con.commit();

        if check == None:
            status.set_text( 'تم تحديث معلومات'

```

```
الموظف بنجاح ' );
```

```
def deleteEmp( widget, data = None ):
    select = tree.get_selection();
    ar = select.get_selected();
    name = ar[ 0 ].get_value( ar[ 1 ], 0
);

    delete = cur.execute( 'DELETE FROM
names WHERE name="' + name + '"' );

    check = con.commit();

    if check == None:
        store.remove( ar[ 1 ] );

con = sql.connect( 'employees' );
cur = con.cursor();

builder = gtk.Builder();
builder.add_from_file( 'gui.glade' );

window = builder.get_object( 'window1' );
tree = builder.get_object( 'treeview1' );
add_emp = builder.get_object( 'button1'
);
update_emp = builder.get_object(
'button2' );
delete_emp = builder.get_object(
'button3' );

signals = { "addNewEmp" : addNewEmp,
```

```

        "updateEmp" : updateEmp,
        "deleteEmp" : deleteEmp,
        "des" : des,
        "delete" : delete}

builder.connect_signals( signals );

# ... #

col1 = gtk.TreeViewColumn( 'الاسم',
gtk.CellRendererText(), text = 0 );
col2 = gtk.TreeViewColumn( 'الراتب',
gtk.CellRendererText(), text = 1 );
col3 = gtk.TreeViewColumn( 'العمر',
gtk.CellRendererText(), text = 2 );

tree.append_column( col1 );
tree.append_column( col2 );
tree.append_column( col3 );

# ... #

store = gtk.ListStore( str, int, int );

tree.set_model( store );

# ... #.

get_names_query = cur.execute( "SELECT *
FROM names" );
names = cur.fetchall();

if len( names ) > 0:

```

```

    k = 0;
    while k < len( names ):
        store.append( [ names[ k ][ 1 ],
names[ k ][ 2 ], names[ k ][ 3 ] ] );

        k += 1;

# ... #

window.show();

gtk.main();

```

هذا و الحمد لله رب العالمين و صلواته و سلامه على خير المرسلين و آله الطاهرين، و ما توفيقى إلا بالله العظيم.

والله من وراء القصد.

الملحق أ : وصلات

برامج تستخدم GTK

GIMP : <http://www.gimp.org>

/AbiWord : <http://www.abisource.com>

/Pidgin : <http://pidgin.im>

/Liferea : <http://liferea.sourceforge.net>

/GNOME : <http://www.gnome.org> مشروع

/Xfce : <http://www.xfce.org> مشروع

/GPE : <http://gpe.linuxtogo.org> مشروع

وصلات دروس

1. درس متكامله بإسم "PyGTK 2.0 Tutorial" على العنوان التالي :
<http://www.pygtk.org/pygtk2tutorial/index.html>
2. درس بإسم "Creating a GUI using PyGTK and Glade" على
العنوان التالي :
<http://www.learningpython.com/2006/05/07/creating-a-gui-using-pygtk-and-glade>
3. درس بإسم "Building an Application with PyGTK and Glade"
على العنوان التالي :
<http://www.learningpython.com/2006/05/30/building-an-application-with-pygtk-and-glade>
4. درس بإسم "A Beginner's Guide to Using pyGTK and Glade"
على العنوان التالي :
<http://www.linuxjournal.com/article/6586>
5. درس بإسم "GTK+ and Glade3 GUI Programming Tutorial"
على العنوان التالي : <http://www.micahcarrick.com/12-24-2007/gtk-glade-tutorial-part-1.html>

6. درس باسم "Using SQLite in Python" على العنوان التالي :
<http://www.devshed.com/c/a/Python/Using-SQLite-in-Python>
7. دروس باسم "PyGTK tutorial" على العنوان التالي :
[/http://zetcode.com/tutorials/pygtktutorial](http://zetcode.com/tutorials/pygtktutorial)

المصادر

1. الدليل الرسمي لـ PyGTK على العنوان التالي :
<http://www.pygtk.org/docs/pygtk/index.html>
2. درس بعنوان "GTK+ 2.0 Tree View Tutorial" على العنوان التالي :
[/http://scentric.net/tutorial](http://scentric.net/tutorial)
3. موقع Glade الرسمي على العنوان التالي :
[/http://glade.gnome.org](http://glade.gnome.org)
4. درس بعنوان "Model-View-Controller Pattern" على العنوان التالي :
<http://www.enode.com/x/markup/tutorial/mvc.html>
5. وثائق PySQLite الموجوده على العنوان التالي :
<http://oss.itsystementwicklung.de/download/pysqlite/doc/sqlite3.html>
6. موقع +GTK الرسمي على العنوان التالي :
[/http://www.gtk.org](http://www.gtk.org)