بناء التطبيقات مع Python

بالإعتماد على GTK و SQLite

الطبعة الثانية

تأليف : محمد قاسم حسين

المحتويات

مقدمة الطبعة الثانية

بسم الله الرحمن الرحيم، و الصلاة و السلام على ا شرف المر سلين سيدنا و مولانا ابا القا سم محمد و على آله الطيبين الطاهرين و رضوان الله على الصحابة المنتجبين اما بعد.

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

طَرحتُ الطبعةَ الأولى من هذا الكتاب في عام 2009، مرّت الأعوام و في فتره من الفترات و لبعض الظروف أُغلقت مدونتي التي كانت تحتوي على رابط تحميل مبا شر لهذا الكتاب، بعد فتره أعدت إفتتاح المدونة و أعدت رفع الكتاب لمن أرا د الإ ستفا دة 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 أنْ تكون أكثر فائدة ووضوحاً.

من أهم التحسينات على الطبعة الثانية هي الإنتقال من إ ستخدام مكتبة Libglade (التي شُرحت في الطبعة الأولى) إلى إ ستخدام مكتبة

GtkBuilder و التي بدأ المبرمجون بالإنتقال إليها، بالإضافة إلى ذلك تم تحديث لقطات الشا شة و إضافة لقطات شا شة جديدة تُوضّح شكل البرامج التي نبنيها في أمثلة الكتاب، كما قمت بتحسين عام للشيفرات بقدر الإمكان و إختبارها على الإ صدار 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 كالعا دة تم إ ستخدام البرامج الحُرّه من أجل إنجازه - 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 مظلّة نظام التشغيل GNU/Linux توزيعة OpenSuSE. شكراً لجهو د 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 التواصل 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 ا شرف المر سلين سيدنا و مولانا ابا القا سم 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 اقلّب بالمواضيع حتى و صلت إلى لغة البايثون، دققت النظر في المجتمع العربي على الانترنت فوجدت أنّ هناكَ كتابان باللغةِ العربية يتحدثان عن هذه اللغه للمبتدئين، فقررت أنْ لا اكتب عن نفس الموضوع الذي سبقوني به، لانه إن حصل ذلك لن يحققَ الكتابُ هدفه، و فيٍ الحقيقه مللنا من كثرة الكتب التي تتحدث عن الا سا سيات فقط خصو صا بالبرمجه، فما اكثر الكتب التي تشرح ا سا سيات 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 ر سوميه بالاعتما د على قواعد البيانات من اجل ا دارة البيانات.

بالنسبه للمصطلحات، فحاولت ترجمتها بقدر الامكان مع الابقاء على المعنى الصحيح و ان كان هذا المعنى ينافي معنى الكلمه حرفياً. و ارجو أنْ لا يلومني البعض على ترجمتي لهذه المصطلحات كما حدث لي من قبل :-)، كما إنّي وضحتُ مع كل مصطلح قمتُ بترجمته الكلمه الا صليه باللغه الانجليزيه، حتى ا سهّل على من ارا د التبحّر و البحث اكثر و اقرّب الصوره لمن لم يتقبل ترجمتي للمصطلح.

هناك نقطه هامه جداً لا بد من ذكرها، و هي أنّ هذا الكتاب يتطلب معرفة سابقه في لغة بايثون، لن يعلمَك الكتاب ما هي البرمجه او ما هي بايثون و كيفية التعامل معها، من الافضل مراجعة الكتب المتخصصه للمبتدئين في هذا المجال، كذلك يفترض هذا الكتاب أنّ لديك إلمام في لغةِ SQL من اجل التعامل مع قواعد البيانات.

هذا الكتاب مجاني حُر يقع تحت رخصة GNU FDL، و تم ا ستخدام البرامج الحرّه 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 مظلّة نظام التشغيل GNU/Linux توزيعة OpenSuSE، ا سمحولي برفع القبعه لمجتمع المصا در 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 أنْ تراسلوني بأسئله حول مواضيع الكتاب لأنها حتماً ستزيد من خبرتي و ستساعدني في الطبعه الثانيه من هذا الكتاب الذي افكّر بوضع ملحق لأسئله القُراء و اجوبتها ان شاء الله :-).

جميع الشيفرات البرمجيه في هذا الكتاب مُجرّبه 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/

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

شكر و تقدير

أشكر مجتمع البرمجيات الحُرّة على جهودهم الجبّارة التي يبذلونها في إنتاج مثل هذه الأدوات الرائعة.

إهداء

إلى والديّ.

الفصل الأول

مدخل إلى GTK

مقدمة

ما هي GTK

GTK عباره عن مكتبه برمجيه لتصميم واجهات مستخدم رسوميه GUI، فبدلًا من بناء برامجنا على شكل سطر اوامر يمكننا بناءه على الشكل الحديث للبرامج عن طريق واجهة مستخدم رسوميه، بمعنى انه يمكن بناء واجهات البرامج ووضع الازرار و مربعات النص و غيرها من الامور مع ما ينا سب برنامجنا، كلمة GTK اختصار للجمله GIMP Toolkit، تقع هذه المكتبة تحت رخصة GNU LGPL و تتميز بكفاءتها مع اللغه العربيه، 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 :-)، تُسمى المكتبه التي تدعم GTK في البايثون بـ PyGTK، الجدير بالذكر ان مكتبة GTK هي المكتبه الا سا سيه التي يستخدمها سطح مكتب GNOME لبناء تطبيقاته وواجهته، بينما يستخدم سطح المكتب KDE مكتبة Qt لبناء الواجهات الرسوميه، و من ابرز البرامج التي تم تصميمها بإستخدام 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 غنو\لينكس و آبل

ماكنتوش و مايكرو سوفت وندوز، هذا يعني امكانية كتابة برنامج واحد ليعمل على الانظمه الثلاث.

نظره سريعه على ماضي GTK

بدأت GTK اساساً 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 رسوميه) و هذا هو سبب تسمية GTK بـ GIMP Toolkit، تعتبر مكتبة GTK احد اجزاء مشروع GNU و تُستخدم الآن بشكل اساسي في نُظم يونكس (مثل غنو\لينكس و FreeBSD و غيرها، و كما ذكرنا مسبقاً انه يعمل على نظم التشغيل الاخرى مثل مايكرو سوفت وندوز).

قبل البدء

اولًا : قبل ان نبدأ بأي شئ لا بد من تثبيت مكتبة PyGTK على حا سوبنا، يمكنك الحصول على هذه المكتبه من موقعها الر سمي .http://www.pygtk.org

ثانياً : في كل ملف Python نو د ا ستخدام PyGTK فيه، لا بد من كتابة الاوامر التاليه :

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

و كما ذكرْت في مقدمة الكتاب، إنّ من يقرأ هذا الكتاب لا بد أنْ يكون مُلماً بلغة Python، و هذا يعني إنّ الشيفره المذكوره في الاعلى مفهومه و ما هي إلا ا ستدعاء لمكتبين، أما السطر الثاني فهو يُلزم مكتبة pygtk على اختيار الا صدار 2.0 فما فوق.

المثال الاول : مرحباً بالعالم

حتى نبدأ بالدراسه العمليه للمفاهيم الاساسيه و التي سوف تساعدنا على بناء التطبيقات يجب أنْ نبدأ بمثال بسيط يستخدم هذه المفاهيم الاساسيه ثم نَلجُ في شرحٍ المفاهيم إستنادا إلى هذا المثال، كالعاده سوف نبدأ ببرنامج "مرحبا بالعالم!" او كما نعرفه بإسم 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 الرسوميه، و بالطبع من ضمن هذه المكتبات مكتبة GTK، الادوات هي كُل ما يُعرض بواجهة البرنامج، مثلًا 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 الرئيسيه، و الزر المكتوب عليه "مرحباً بالعالم".

الادوات مع الـ PyGTK

البدايه

في PyGTK لكل اداة من الادوات التي يقدمها فئة (Class) خاصة 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 اولًا 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 ا سمه window، و ا ستخدمنا الفئة gtk.Window من اجل إنشاء النافذه الرئيسيه و كلفنا المتغير window بها و بالتالي أ صبح كائناً (Object) يمكننا من خلاله التحكم بالنافذه (لا بد من الملاحظه ان تكون قد درست الـ OOP مع الـ Python حتى تتمكن من ا ستيعاب ما يُشرح)، و كما ذكرنا مسبقاً إنّ كل ا داة في GTK يكون لها فئة خا صة بها، هذا يعني أنّ لدينا العديد من الفئات غير الفئة gtk.Window، مثلاً 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 الا سا سيه منها حتى نتعلم طريقة الكتابه، و بعدها يمكنك التبحر عن طريق قراءة الدليل الر سمي الخاص بـ PyGTK و الذي يشرح جميع الفئات المتوفره.

على كل حال نعو د لمثالنا، كما أ سلفنا إنّ المتغير window أ صبحَ كائناً (Object)، يمكننا من خلال هذا الكائن التحكم بالنافذه 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 أحدى الدوال التي تُقدمها الفئة gtk.Window و التي تساعدك على التحكم في الاداة التي قمت بإنشاءها، تقوم هذه الداله بتغيير عنوان النافذه إلى محتوى البارامتر المُمرر لها و في حالتنا هذه فإن القيمة هي "مرحباً بالعالم"، بالطبع هناك العشرات من الدوال التي يقدمها الصنف gtk.Window يمكنك القراءه عنها في دليل PyGTK الرسمي، لنأخذ مثالاً سريعاً على احد هذه الدوال، الداله resize تقوم بتغيير حجم النافذه، قم بإضافة السطر التالي اسفل السطر المدروس :

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

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

الآن تجاهل السطرين التاليين لأنّنا 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"

ننتقل إلى السطر التالي وهو :

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

بعدما قمنا "بإنشاء" الزر، و قمنا بالتعديل عليه بما ينا سب برنامجنا (مثل اختيار النص الذي يظهر على الزر و غيره من هذه الامور) لا بد من "عرض" الزر و هذا يتم بإ ستخدام الداله show، نستخدم الداله show مع جميع الا دوات تقريباً حتى يتم عرض الا داة بعد إنشاءها.

ننتقل الآن إلى السطر التالي :

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

بعدما 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 ا ستخدام هذه الداله إلا مره واحده فقط في برنامجنا، اي اننا لا نستطيع إلا اضافة ا داة واحد فقط إلى النافذه الرئيسيه، قد تتسائل في هذه الحاله كيف يمكننا وضع اكثر من ا داة في نافذتنا الرئيسيه؟

لنفرض إننا نريد تصميم برنامج لتحويل القيا سات، في هذه الحاله سوف نحتاج إلى مجموعه من العنا صر، مثلاً مربع نصوص لإ ستقبال الرقم

المطلوب تحويله، زر ليضغط عليه المستخدم بعد كتابة المعلومات، سطر نصّي لطباعة النتائج، في هذه الحاله كيف يمكنني اضافة جميع هذه الادوات على نافذتي الرئيسيه؟ و في الحقيقه لا يمكنني استدعاء دالة الاضافه add إلا مره واحده فقط!. الجواب على هذا السؤال هو بإستخدام 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، الداله show تقوم بعرض الاداة بعد إنشاءها 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 الرئيسيه تعتبر من ضمن الادوات و بالتالي لابد من عرضها بعد إنشاءها، و هذا ما يتم في السطر الذي ندرسه الآن.

ننتقل 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 "مرحباً 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 واحدة في برنامجنا، ذكرنا في ذاك الموضع إنّ الحلَ مع الصناديق.

الصناديق HBox و VBox هي ادوات توفرها PyGTK، تُعتبر الصناديق من اهم الادوات التي سوف نستخدمها في جميع برامجنا التي تحتوي على أكثر من اداة، لِأوضِّح الفكرةَ أكثر : في PyGTK يجب أنْ تكون كل اداة في صندوق لوحده، مثلاً لنفرض اننا نريدُ تصميم واجهه تحتوي على مربع نص و زر واحد، افتراضياً في النافذه التي نُنشئها في PyGTK يمكننا إضافة اداة واحده فقط هذا يعني اننا لن نتمكن من اضافة اداة اخرى، لحل هذه المشكله لا بد من اضافة 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 الصنا ديق في PyGTK، النوع الاول هو HBox و هذا النوع يقوم بإنشاء الصنا ديق بشكل افقي، اما النوع الثاني هو VBox و الذي يقوم بإنشاء الصنا ديق بشكل عمو دي، حسناً الآن ا ذا ار دنا وضع الزر بجانب صندوق النص سوف نقوم بإ ستخدام HBox، اما ا ذا ار دنا وضع الزر في ا سفل مربع النص (او العكس) سوف نستخدم VBox، الجدير بالذكر انه يمكننا وضع صنا ديق داخل صنا ديق.

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 متساويه لما يقع داخل الصندوق ا ذا كانت True.

البارامتر الثاني spacing و هي المسافه 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 مسبقاً.

إستخدام الصناديق عملياً

نعود لمثالنا "مرحباً 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، اولًا :

```
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 إننا لم نمرر أي بارامترات لأننا نُريد القيم الإفتراضية 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.

ننتقل 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 أضافة الاداة hello_world_button داخل الصندوق، سوف نستخدم الداله pack_start دائماً من اجل إضافة الادوات داخل الصناديق، و على اختلاف نوع الصندوق سواء كان صندوق VBox او HBox فلا يوجد فرق فالداله نفسها.

ننتقل 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 شئ جديد هنا لاننا شرحنا معنى هذه الا سطر مسبقاً، هذه الا سطر 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.

ننتقل إلى آخر سطر جديد قمنا بإضافته :

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

في هذا السطر أظهرنا الصندوق الذي وضعنا ا دواتنا فيه، لاحظ انه يمكننا إظهار الا داة في اي مكان، مثلاً لا يوجد مشكلة لو قمنا بإظهار الصندوق بعد إنشاءه مبا شره في مثالنا هذا، ببساطه يمكننا ا ستدعاء الداله show و إظهار ا داتنا في اي مكان، المهم ان ننشئ الا داة اولاً.

الاشارات و الدوال Callback

مقدمه

في بادئ الامر درسنا مفهوم الادوات (Widget) و كما تعلم إنّه مفهومٌ مهمٌ جداً، المفهوم الآخر و الذي يجب إستيعابه بشكلٍ جيّد هو "الاشارات" (Signals).

تخيل معي ان برنامجنا يحتوي على زر مكتوب عليه "اضغط هنا" عندما يضغطُ المستخدمُ على الزر لن يحدثَ اي شئ! لاننا وضعنا الزر ضمن النافذه لا اكثر، اذا اردنا من الزر أنْ يقومَ بعملٍ مفيدٍ سوف نستخدم شيئين هما الاشارات (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.

الاشارات مع PyGTK

هل تتذكر السطرين الذين تجاهلناهما عندما بدأنا بشرح مثالنا الاول "مرحباً بالعالم"؟ نعم لا بد و إنّك تتذكرهما :-)

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

هذان السطران تطبيقٌ للاشارات، مع اي اداة ننشؤها يمكننا استخدام الداله connect حتى نجعل الاداة تقوم بعمل مفيد، في مثالنا الحالي قمنا بربط النافذه بحدثين، الحدث الاول يُسمى delete_event و الثاني يُسمى destroy، نستخدم هذين الحدثين مع النافذه الرئيسيه فقط لانهما يساعدان على إغلاق البرنامج عندما يطلب المستخدم ذلك، عندما يتم تنفيذ الحدث الاول تُستدعى الداله delete_event و التي عرّفناها مسبقاً، و عندما يتم تنفيذ الحدث الثاني تُستدعى الداله 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 الاقل بارامترين كما تلاحظ في تعريفنا لهاتين الدالتين، حتى تتضح الصوره بشكلٍ اكبر سنطور مثال "مرحبا بالعالم"، بحيث اذا ضغطنا على زر "مرحبا بالعالم" تظهر لنا نافذه 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 جديده.

```
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 إنّ 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 اضفنا إليها نص للقراءه "هذا برنامجي الاول مع PyGTK"، الشئ الجديد في هذه الداله هي الاداة Label والتي لم نتعامل معها مسبقاً، تقوم هذه الداله بطباعة النص الذي تقوم بوضعه كنص للقراءه.

هناك شئ هام ذكرناه مسبقاً و سوف نعيد ذكره هنا بشئ من الإسهاب، جميع الدوال التي صنفناها من النوع callback يجب أنْ تستقبلَ بارامترين على الاقل، و هذا الذي تلاحظه عندما عرّفنا 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 هيٍ hello_world_button (الزر المكتوب عليه مرحباً بالعالم)، في هذه

الحاله يكون البارامتر widget هو نفسه hello_world_button، و بالتالي يمكنك ا ستخدام جميع الدوال التي يقدمها الصنف gtk.Button داخل الداله the_new_window عن طريق البارامتر widget.

ننتقل إلى البارامتر الثاني وهو data، تخيل معي اننا نريد تمرير بعض البيانات إلى الداله the_new_window، كيف يمكننا ذلك؟ نعو د 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، مثلاً :

```
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 الداله the_new_window و قم بطباعة محتوى البارامتر data في نهايتها، و جرّب المثالين السابقين، في اول مثال سوف تطبع كلمة test و في المثال الثاني سوف تطبع السطر ['One','Two','Three']، بالطبع هذا الاسلوبٍ سيفيدك كثيراً اذا كنت لا تستخدم OOP في كتابة برامجك، لانك احياناً 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 الحراره

سوف نقوم الآن بإستخدام ما تعلمناه و نبني برنامجاً جديداً، يأخذ هذا البرنامج درجة الحراره بالسيليزي و يحولُها إلى الفهرنهايتي، هذا يعني إنّ برنامجنا يحتوي على مربع نص لأخذ القيمه السيليزيه من المستخدم، و كذلك سوف نحتاج إلى زر يضغط المستخدم عليه عند الانتهاء من وضع القيمه المُراد تحويلها، و اخيراً نحتاج إلى نص (Label) لطباعة الناتج، يُفترض أن يكون شكل برنامجنا كالتالي :

لنبدأ اولاً بكتابة الاسطر التي اتفقنا على كتابتها في بداية الكتاب :

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

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

```
import gtk;
```

نبني الآن النافذه 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 اولاً، 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 بإسم 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 تُستدعى داله معينه تقوم بالعمليه الحسابيه 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 تُستدعى عندما يتم تنفيذ الحدث، وفي حالتنا هذه الحدث هو الضغط على الزر.
البارامتر الثالث وهو ما جلعني اتوقف هنا، هل تتذكر 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 بإسم data، لابد و انك تعرف هذه المعلومات و لكن وجب التذكير، قد تتسائل 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 مصفوفه تحتوي على عنصرين، العنصر الاول هو

الكائن status_label و الذي يتيح لنا التحكم بأداة النص (Label)، اما العنصر الثاني هو الكائن text_entry و الذي يتيح لنا التحكم في اداة مربع النص، و بالطبع البارامتر widget يتيح لنا التحكم بالزر.

مررنا الكائن status_label على الداله clickEvent حتى نتمكن من تغيير النص الموجود في اداة النص (Label) من داخل الداله، أما الكائن text_entry مررنا حتى نتمكن من أخذ القيمة التي يكتبها المستخدم في مربع النص، هذا يعني إنّنا نريد الوصول إلى هذه الادوات من داخل الداله، و بالتالي مررنا كائنات هذه الادوات كبارامتر للداله.

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 نخزّن القيم في متغيرات جديده ذات اسم اوضح :

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

نأخذ الآن القيمه الموجوده في داخل مربع النص بإستخدام الداله get_text و التي توفرها الفئة gtk.Entry :

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

نحوّل هذه القيمه من نوع string إلى نوع float :

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

نُجري عمليتنا الحسابيه :

```
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 بإستخدام 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 بدلًا من تمرير الادوات التي نريد التحكم بها من داخل الداله فبدلًا من تمريرها كبارامترات للداله clickEvent يمكننا استخدام الكلمه global للو صول إلى كائنات هذه الادوات بدون الحاجه إلى تمريرها كبارامترات، و بالتالي 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 شئ، ما رأيك أليست اسهل و اكثرا اختصاراً؟ :-)

سنستخدم global من الآن فصاعداً. لاحظ إنه لا يمكن إستخدام 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 تسهّل على مستخدم البرنامج؟ :-)

وفقاً للشيفرة 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 الإختيار 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 الإختيار و لكنني هنا سأربطه بمربع النص هنا، لاحظ إنه يمكننا ربطه في النافذه الرئيسيه 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 لابد من توضيحها، بعض الأحداث مثل key_release_event تُحمل 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 تهمنا لأنها ستكون فيما بعد القيمة السيليزية التي سنحولها إلى فهرنهايتيه.

لنبدأ بالعمل الحقيقي الآن :-)، نُعرّف 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 إعا دة 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 كُل 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 بدلاً من الضغط على زر موافق في البرنامج، و

سيظهر الناتج :-)

الشيفرة 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 مثالاً آخراً، هذا المثال عباره عن آله حا سبه، تقوم بالعمليات الا سا سيه، جمع، طرح، ضرب، قسمه.

تحتوي واجهة البرنامج على ثلاث مربعات نص يُكتب في المربع الاول العدد الاول و تُكتب العمليه المطلوب تنفيذها في المربع الثاني و اخيراً يُكتب الرقم الثاني في المربع الاخير، بالاضافه إلى ذلك يكون هناك زراً لإظهار الناتج و يكون هناك نصاً (Label) يظهر فيه الناتج بعد الضغط على الزر، شكل البرنامج كالتالي :

كما تعودنا نكتب اولاً الشيفرات الا سا سيه 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، و جميعها بجانب بعضها البعض، و في اسفلها جميعها يكون هناك زر و اسفل الزر هناك نص تظهر فيه النتائج، في هذه الحاله 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 عمو ديه، الخانه الاولى تحتوي على مربعات النص، الخانه الثانيه تحتوي على الزر و الخانه الثالثه تحتوي على النص (Label)، المشكله هنا أنّ الخانه الواحده لا تستوعب إلا اداة واحده، و بما أنّ نوع الصندوق VBox اذاً سيتم وضع كل اداة جديده يتم اضافتها إلى الصندوق في اسفل الاداة التي تسبقها.

إذاً 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 اخيراً نضيف الصندوق HBox إلى الصندوق الرئيسي من نوعٍ VBox في الخانه الاولى، هذه الطريقه هي الطريقه المُستخدمه دائماً 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 اتفقنا، في الخانه الاولى سنضع مربعات النص الثلاثه بجانب بعضهن، ووفقاً لما شرحناه بالاعلى سننشئ صندوقاً جديداً 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 بدلاً 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 نُظهره :

```
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 اولاً 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 اخيراً نُظهر الصندوق الرئيسي، النافذه، و ننا دي الداله الا سا سيه :

```
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 الزرُ داله معينه تقوم بالعمليه الرياضيه ثم تطبع الناتج على النص (Label) هذا كل شئ.

ا ذاً نبدأ مع الزر، نربط الزر بالحدث clicked ليتم إستدعاء داله ا سمها doMath عن طريق السطر التالي و الذي سوفه نضيفه فوق السطر الذي يضع الزر في الصندوق الرئيسي :

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

نُعرّف 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 نُظهر 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 الا صول و جميع المعلومات المطلوبه متوفره، نبدأ بالعمليه الحسابيه، في با دئ الامر نحوّل نوع كل من المتغيرين operand1 و operand2 إلى integer حتى يكون ناتج العمليه الحسابيه صحيحاً لأنها حالياً عباره عن نصوص (أو سلا سل Strings) :

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

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

نتحقق الآن من النص الذي ا دخله المستخدم في المربع الثاني، هل هو + او - او * او \ ثم نقوم بالعمليه المطلوبه وفقاً لمحتوى المربع الثاني، اذا كان محتوى المربع الثاني لا يساوي ايا من القيم المذكوره سنطبع ر ساله 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( "العمليه التي قمت بإختيارها غير صحيحه" );
    return False;
```

و اخيراً نطبع الناتج للمستخدم بعد تحويله إلى نص :

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

آلتنا 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 بدلًا من إجباره على الضغط على زر "أظهر الناتج" في البرنامج.

كما تعلمنا سابقاً، يجب ربط أحدى 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، لما ذا؟ لأننا لا نعلم أين قد يضغط المستخدم على مفتاح الإدخال، غالباً سيضغط عليه بعدما يكتب القيمة في المربع الثالث لأنها آخر قيمة سيُدخلها، لكن ما ذا 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 الفأره و هكذا إختفت الميزة التي أضفناها! :-)

لنبدأ بربط الأحداث، نُضيف 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 );
```

الشيفرة 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( "
العمليه التي قمت بإختيارها غير صحيحه" );
            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 الخا صه بواجهة البرنامج عمليه مملة و على العكس قد يكون هناك ا شخاص يتمتعون عندما يكتبون شيفرة واجهة البرنامج، و لكن لنفكر بالموضوع قليلاً، ا ذا قررنا كتابة برنامج ضخم با ستخدام PyGTK فإننا سنستغرق الكثير من الوقت من اجل كتابة الشيفره المصدريه الخا صه بالواجهه، و قد تكون عملية الاضافه على الواجهه في ما بعد عمليه متعبه و طويله، لحسن الحظ هناك حل لهذه المشكله و هو برنامج Glade.

يسهّل 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 الر سوميه.

إلى وقت إ صدار الطبعة الثانية من هذا الكتاب (سنة 2013) فإنّ الإ صدارات الحالية من Glade هي الثالثة، و أحدث إ صدار هو 3.12.

الجيل الثالث من Glade كان اعادة كتابه من الصفر، من وجهة نظري ان واجهة الجيل الثالث افضل بكثير من واجهة الجيل الثاني.

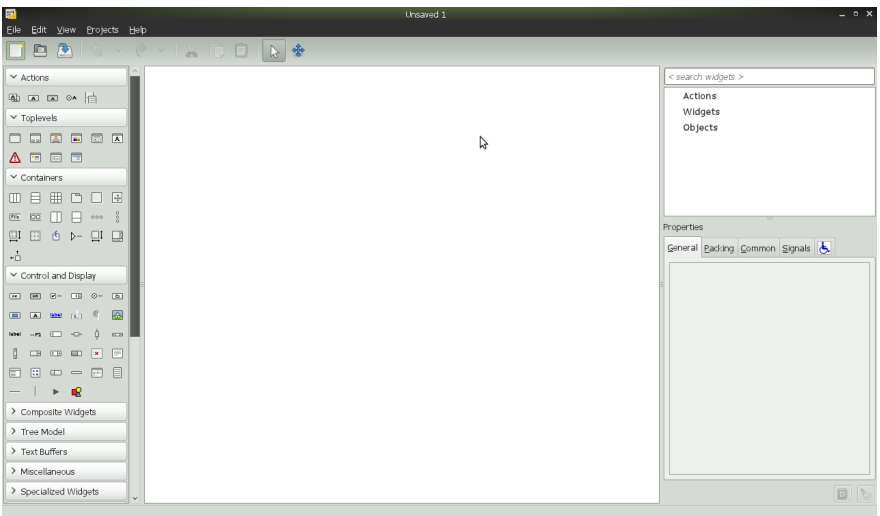
يُمكن التعامل مع ملف XML الناتج من برنامج Glade في النص البرمجي كما سنرى بعد قليل، في الطبعة الأولى (سنة 2009) من هذا الكتاب تم شرح المكتبة البرمجية Libglade و هي المسؤولة عن التعامل مع ملفات XML الناتجة عن برنامج Glade في النص البرمجي، في عام 2007 صدَرت النسخة 2.12 من مكتبة GTK و جاءت معها مكتبة GtkBuilder، 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 بالإنتقال من Libglade إلى GtkBuilder، و بالتالي سيكون الشرح القادم قائم على مكتبة GtkBuilder بدلاً من Libglade.

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

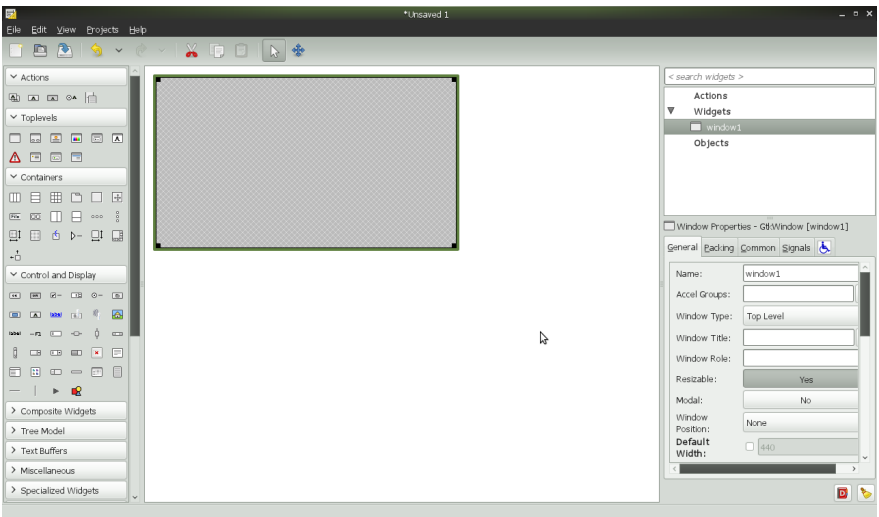
التصميم مع Glade

لقد تعلمنا في الاقسام السابقه كيف نكتب شيفرة واجهة المستخدم، عملية تصميم واجهة المستخدم مع Glade اسهل بكثير، بمجرد رؤيتك لواجهة البرنامج سوف ترى إنّ العمليةَ بسيطةٌ، لنلقي نظره على الصوره التاليه :



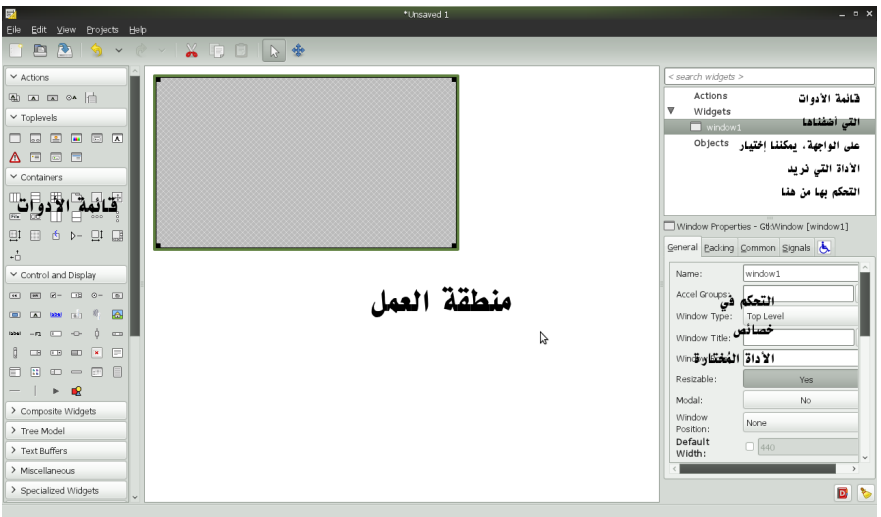
هذا هو برنامجٍ Glade الذي سنستخدمه لتصميم واجهاتنا من الآن فصاعداً، حسنا لنرى الآن كيف يمكننا استخدام هذا البرنامج في تصميم واجهتنا، نلاحظ على الجانب الايسر قائمه تحتوي على الادوات مثل

النافذه و الصنا ديق و مربعات النص و غيرها الكثير من الا دوات التي لم نستخدمها مسبقاً، هذه هي جميع الا دوات التي تقدمها GTK، كما تعو دنا دائماً في البدايه نحن بحاجه إلى نافذه حتى نضيف عليها ا دوات البرنامج، ستجد النافذه في قائمة Toplevels و ا سمها Window، الصوره التاليه توضح لك اننا اضفنا نافذه إلى واجهتنا التي نبنيها :



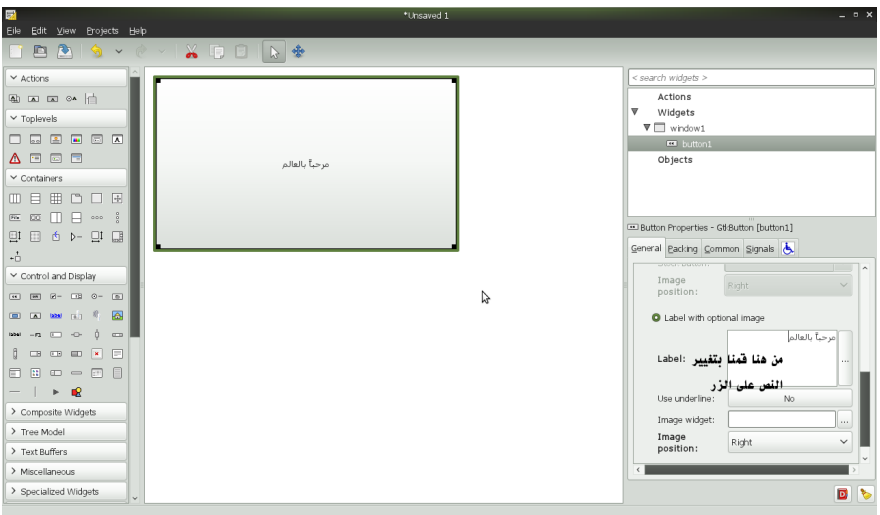
لاحظ الفرق بعد اضافة النافذه، في المنتصف ظهرت النافذه التي اضفناها و التي سوف نضيف فوقها الا دوات، ا ذاً الجزء الذي يقع في المنتصف هو مكان العمل، اما جهة اليمين نلاحظ ان هناك قائمه في الاعلى، هذه القائمه

تحتوي على جميع الادوات التي اضفناها في مشروعنا هذا، اما اسفل هذه القائمه نجد صندوقاً 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 "مرحباً بالعالم" هذا كل شئ!

هيا لنبدأ نضيف اولًا النافذه تجدها كما ذكرنا تحت القسم Toplevels بإسم Window، لأننا سنضيف اداة واحده فقط و بالتالي لسنا بحاجه إلى استخدام الصناديق، نضيف الزر إلى النافذه مباشره، بعد إضافة الزر نذهب إلى خصائصه لنغيرها (يمين الشاشه في الاسفل)، يجب ان يكون التبويت المُختار هو General نجد حقل اسمه label و بجانبه مربع نص كبير، نغيّر النص في هذا المربع إلى "مرحباً بالعالم"، يجب ان يظهر لك شئ مشابه للصوره التاليه (بالمناسبه لا تقلق بشأن حجم النافذه) :



نخزّن الآن واجهتنا البسيطه، لاحظ إنّك عندما تطلب من Glade تخزين الملف، سيظهر لك في أسفل شاشة التخزين خيارين هما GtkBuilder و Libglade، دائماً إختر GtkBuilder فكما وضّحت مسبقاً فإننا سنستخدم مكتبة GtkBuilder البرمجية من أجل التعامل مع ملف XML الناتج من Glade.

من اجل الترتيب من الافضل إنشاء مجلد خاص ببرنامجنا لنخزّن فيه الملف، لنسمي ملفنا بإسم gui، بعد التخزين يجب ان يكون لدينا في المجلد ملف بإسم 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 بإستخدام 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 GtkBuiler حتى نتمكن من تحميل ملف Glade و التعامل معه في النص البرمجي :

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

سيكون تعاملنا الآن مع عنا صر الواجهة الر سومية 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 المُراد 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 "مرحباً بالعالم".

في السابق عندما كنّا نكتب واجهاتنا الر سوميه عن طريق الاسطر البرمجيه 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 ا ستخدام الداله get_object، تأخذ هذه الداله ا سم الاداة التي تريد التحكم بها كبارامتر.

ارجع إلى برنامج Glade و شغّل الملف gui.glade، اختر النافذه و انظر في خصائصها، انظر الخا صيه Name سوف تجد ان المكتوب في المربع الواقع بجانب Name هو window1 هذا هو الا سم الذي سوف نمرره إلى الداله get_object، كذلك بالنسبه للزر سوف تجد ان ا سمه button1، بالطبع يمكنك تغيير هذه الا سماء إلى ا سماء انسب، من وجهة نظري الإبقاء على الا سماء الافتراضيه التي يضعها Glade تلقائياً مثل window1 و button1 و button2 عا دة سيئة و سوف يظهر ذلك بوضوح عندما تصمم برنامج ذو واجهه معقده، لذلك من الافضل وضع إ سم ذو معنى واضح لكل ا داة حيث يُبين هذا الإ سم وظيفة الأ داة في الواجهة الر سومية.

غيّر الآن ا سم النافذه إلى main_window، و ا سم الزر إلى hello_world_button و احفظ التعديلات، حان الوقت لنستخدم الداله get_object :-) :

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

نقوم بإظهار الا دوات عن طريق الداله show :

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

نستدعي الداله الا سا سيه :

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

حسناً لا بد و انك تعلم اننا الآن يمكننا ا ستخدام الكائنين window و button من اجل التحكم بالا داتين و بالتالي يمكننا ا ستخدام جميع الدوال التي تقدمها الفئة gtk.Window او الفئة gtk.Button، مثلا ا ذا ار دنا تغيير عنوان النافذه يمكننا ا ستخدام الداله set_title و هكذا.

الآن ا صبح من الواضح كيفية ا ستخدام الملفات التي يولّدها برنامج Glade في داخل برامجنا، سوف ننتقل إلى القسم التالي و نتعلم كيف يمكننا التعامل مع الا شارات.

الشيفره 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 بدالة الإغلاق :

اولًا 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 شئ!

الشيفره 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 السابقه و يفترض بك أنْ تكون قد فهمته بشكل جيّد.

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

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

بجانب الكلمه clicked سوف نجد مربعٍ نص يمكننا الكتابه فيه، مكتوب اعلى هذا المربع Handler، نكتب دائماً في هذا المربع اسماً مميزاً، شخصياً افضّل كتابة اسم الداله التي سوف تعالج الحدث، يمكننا تغيير الاسم إلى اي شئ آخر و لكن حتى لا نزيد الاسماء في برنامجنا نستخدم اسماً موحّداً، سنستخدم هذا الاسم في النص البرمجي بحيث نضعه في

مصفوفه (او قاموس) و يكون الا سم مفتاح المُدخل و تكون قيمته هي الداله التي يجب ا ستدعاءها، في حالتنا هذه الداله التي سوف تعالج الحدث هي buttonClicked و التي كتبناها منذ قليل، ا ذا قم بوضع buttonClicked في Handler، احفظ المشروع و لننتقل إلى النص البرمجي.

سوف نعدّل على المثال السابق و نحوّل الا شارات فيه من الطريقه العا ديه إلى طريقة Glade، و بالتالي يجب علينا حذف الأسطر التاليه من النص البرمجي :

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

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

ننشئ الآن المصفوفه التي اتفقنا على إنشائها، تحتوي هذه المصفوفه على مُدخل واحد فقط، له مفتاح بإ سم buttonClicked، و قيمة هذا المفتاح هي الداله ButtonClicked، لا تخلط بين الاثنين، المفتاح هو الا سم الذي حد دناه في Glade و ليس من الضروري ان يكون بنفس إ سم الداله، اما الا سم الثاني فهو ا سم الداله نفسها، سوف نسمّي هذه

المصفوفه بإسم 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 الرئيسيه، انتقل إلى التبويب Signals.

كما تعودنا سوف نربط النافذه بالحدثين delete_event والذي تجده في GtkWidget، كما اتفقنا سوف نضع اسم الـ Handler بنفس اسم

الداله و دالتنا هنا هي delete_event، الحدث الثاني هو destroy تجده في GtkObject، سوف نسمي الـ Handler هنا بإسم destroy، احفظ العمل و انتقل إلى النص البرمجي.

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

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

builder.connect_signals( signals );
```

جرّب البرنامج الآن و ستجده يعمل وفقاً للا صول :-)

الشيفره 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 الحراره إلى Glade

بالطبعٍ 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 بإستخدام Glade، برنامج تحويل الحراره يستخدم الصنا ديق، و يستخدم صندوق من نوع VBox و بالتالي أول شيء نفعله بعد إضافة النافذه الرئيسيه هو اضافة صندوق من نوع VBox إلى النافذه، ستجد الصندوق VBox تحت التصنيف Containers تحت ا سم Vertical Box.

عندما تضع الصندوق على النافذه سيسألك Glade : كم عد د العنا صر المطلوبه؟ و كما تعلم إننا نريد إضافة ثلاث أ دوات واحده أ سفل الاخرى، و بالتالي سوف نحتاج إلى 3 عنا صر، بعد اختيارك للعدد 3 و الضغط على موافق تجد إنّ النافذه تقسمت إلى 3 اقسام.

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

نختار الزر و نذهب إلى خصائصه، و نغيّر قيمة الخا صيّه Label إلى "موافق"، بعدها نختار النص (Label) و نذهب إلى خصائصه و نحذف قيمة الخا صيّه Label، نختار الآن النافذه الرئيسيه و نذهب إلى خصائصها

و نضع قيمة الخا صيّه Window Title كالتالي : "برنامج التحويل".

نقوم الآن بعمليات الربط، لنربط النافذه الرئيسيه بالحدثين الذين اعتدنا عليهما و هما delete-event و نضع ا سم delete_event للـ Handler و الحدث destroy و نضع ا سم destroy للـ Handler.

بعدها نضيف الحدث clicked للزر الذي اضفناه، و ليكن ا سم Handler هذا الزر هو clickEvent بنفس ا سم الداله المكتوبه مسبقاً.

أما بالنسبة لمربّع النص فنضيف له الحدث key-release-event و الذي ستجده ضمن قائمة GtkWidget، ليكن إ سم الـ Handler هو keyEvent، هذا كل ما نحتاجه من Glade، خزّن الملف و ليكن ا سمه gui.glade

ننتقل الآن إلى النص البرمجي، أ سفل الداله clickEvent مبا شره 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 أنّنا بحاجه إلى اداتين فقط هما اللتان سنتحكم بهما، الاداة الاولى هي مربع النص، و الثانيه هي النص (Label)، و بالتالي سنستخدم الداله get_object مع هاتين الاداتين 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 الرئيسيه :

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

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

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

هذا كل شئ! :-)

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 الحا سبه 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( "
العمليه التي قمت بإختيارها غير صحيحه" );
                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( "
العمليه التي قمت بإختيارها غير صحيحه" );
            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 الآله الحاسبه حتى نصممها مع Glade، كانت تنقسم إلى ثلاث اقسام، القسم الاول 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 بحاجه إلى صندوق داخل هذا القسم و هذا الصندوق من نوع HBox تجده في القسم Containers بإسم Horizontal Box، و نختار الرقم 3 لاننا بحاجه 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 زراً جديداً، اخيراً نضيف النص (Label) في آخر قسم.

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

نغيّر النص في اعلى الزر إلى "اظهر الناتج" و نربط الزر بالحدث clicked و نسمي الـ Handler بإسم الداله doMath.

نحذف الآن النص الموجود في اداة النص.

نربط مربعات النص الثلاث بالحدث key-release-event و نسمّي الـ Handler بـ keyEvent

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

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

نستخدم الداله get_object مع مربعات النص الثلاثه و نص اظهار النتائج بالاضافه إلى النافذه الرئيسيه :

```
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' );
```

نُنشؤ مصفوفة الإشارات و نربطها بإستخدام 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 شئ! 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( "
العمليه التي قمت بإختيارها غير صحيحه" );
                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 مقارنه بالشيفره 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 التخلص منها، مثلاً البرامج التي تُستخدم في الشركات و التي تديرُ الشركةُ من خلالها بياناتَ الموظفين، كذلك برامج القواميس التي تعطي معاني الكلمات، تستخدم هذه البرامج مُحركات قواعد بيانات لتخزين البيانات و التعامل معها، تقدّم لغة Python العديد من مُحرّكات قواعد البيانات، و لكننا سوف نستخدم المُحرّك SQLite نظراً لبساطته و سهولته، يستخدم هذا المُحرّك لغة SQL من اجل التعامل مع البيانات، أفترض هنا ان لديك إلمام بلغة SQL، اذا لم تكن تعرف هذه اللغه انصحك بمراجعة الكتب المُتخصصه او مواقع الانترنت فهي مليئه بدروس SQL التي تشرح التفاصيل، استخدم اي محرّك بحث و سوف تجد العديد من المواقع على الشبكه.

بالطبع قبل البدأ لابد من تثبيت مكتبة SQLite على حاسوبك، سوف تحتاج إلى المكتبه PySQLite فراجع موقعها للحصول عليها و على كيفية تثبيتها.

الدوال الاساسيه

تقدّم المكتبه PySQLite مجموعه من الدوال، و لكننا لن نحتاجها جميعها، سوف نستخدم بعض الدوال الا سا سيه التي يحتاجها اي برنامج يعتمد على PySQLite، يمكنك كالعا ده القراءه فيما بعد اكثر حول الدوال التي تقدمها PySQLite و قد تجد دوال تهمك.

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، ا ذا لم تجد الداله connect ملف قاعدة البيانات في المسار المطلوب ستقوم بإنشاءه.

الداله الثانيه هي cursor وهي داله ا سا سيه جداً، حيث تأخذ مؤ شر SQLite و تُعيده إلى متغير، سوف نستخدم هذا المتغير فيما بعد في جميع العمليات الاخرى، مثل عملية إنشاء الا ستعلامات او ا خذ ناتج عملية الا ستعلامات، و هذا مثال لا ستخدام هاتين الدالتين :

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

اولًا نستدعي مكتبة SQLite في السطر الاول، في السطر الثاني نتصل بقاعدة البيانات التي ا سمها DB، بما اننا لم نمرر مسار معين و قمنا بوضع ا سم قاعدة البيانات فحسب هذا يعني ان قاعدة البيانات DB مُخزنه في نفس المجلد الذي يحتوي على ملف بايثون، السطر الاخير يأخذ المؤ شر و يخزنه في المتغير cur، بعدها سنستخدم المتغير cur دائماً من اجل العمليات الاخرى.

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

الداله الثانيه وهي fetchall نستخدمها دائماً بعد عمل ا ستعلام من نوع SELECT، بحيث نحصل على مصفوفه تحتوي على النتائج.

اخيراً الداله commit و نستخدمها اذا قمنا بأي عملية تغيير في قاعدة البيانات، مثلاً اضفنا بيانات او حدّثنا بيانات او انشأنا جدولاً جديداً. 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

ا صبح لدينا الآن إلمام بكتابة برامج ذات واجهه ر سوميه بإ ستخدام GTK، و لدينا إلمام مسبق بلغة SQL، و تعرفنا على الدوال الا سا سيه التي تخص مكتبة SQLite، ا ذاً نحن الآن جاهزون لكتابة برنامج ر سومي يعتمد على قواعد البيانات لتخزين معلوماته.

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

نبدأ اولًا بتخطيط جداول قواعد البيانات، نحتاج إلى جدول وا حد فحسب، نخزّن في هذا الجدول الا سماء و ا سم هذا الجدول هو names، اما بالنسبه للحقول نحتاج إلى حقلين، الحقل الاول هو id و الذي يخزّن رقم تعريفي مميز، امّا الحقل الثاني هو name وهو الا سم الذي نو د تخزينه.

الآن لنتحد ث عن واجهة البرنامج، لاننا نريد البرنامج بسيطاً لن يكون هناك عدّة نوافذ، اعني اننا لن نستخدمَ نافذة منفصلة من اجل إضافة ا سم جديد،

و لكن سنستخدم النافذه الرئيسيه و نضع فيها قائمة الاسماء و في اسفلها نضع الادوات اللازمه لإضافة اسم جديد إلى قواعد البيانات.

هذا يعني اننا سوف نقسّم واجهة البرنامج إلى خمس اقسام بإستخدام صندوق من نوع VBox، القسم الاول يحتوي على قائمه من نوع TreeView، القسم الثاني يحتوي على نص (Label) و نغيره 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 الحاله، مثلاً اذا نجحت الاضافه يتم عرض ذلك في هذا النص، يُفترض أن تكون واجهة البرنامج كالتالي :

صممها ثم قم بتخزين الملف بإسم gui، لا ننسى الآن ربط النافذه بالدالتين الاساسيتين، للحدث destroy نستخدم داله بإسم des اما الحدث 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' );
```

مناداة المكتبات و كتابة الدالتين delete و des 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 الثلاثه.

```
window = builder.get_object( 'window1' );
tree = 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 لابد من عمل هام جداً، وهو إنشاء قاعدة البيانات و الجدول الذي سوف نستخدمه، من الأفضل إنشاء ملف برمجي جديد يحتوي على شيفرة إنشاء الجدول، نسمّي هذا الملف بإسم database_create.py و يكون محتواه كالتالي، نستدعي اولاً مكتبة SQLite :

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

from pysqlite2 import dbapi2 as sql;
```

ثم نتصل بقاعدة بيانات إسمها sqlite_db بإستخدام الداله connect المشروحه سلفاً :

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

نخزّن المؤشر في المتغير cur :

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

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

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

هذا كل شئ، سوف يكون الملف كالتالي :

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

from pysqlite2 import dbapi2 as sql;

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

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

نخزّن الملف بإسم database_create.py و نقوم بتشغيله، بعد تشغيل الملف بنجاح نلاحظ أنّ هناك ملف جديد إسمه 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 الداله addNewName يجب أنْ نتصلَ بقاعدة البيانات كما فعلنا تماماً في الملف database_create.py، نذهب إلى الملف main.py و قبل السطر :

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

وهو السطر الخاص بإنشاء كائن Builder، نضيف السطرين الخاصّين 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 الداله addNewName، تقوم هذه الداله بأخذ المحتوى الموجود داخل مربع النص، و التحقق اذا كان المحتوى فارغاً (أي ان مستخدم البرنامج لم يكتب شئاً داخل مربع النص و ضغط على الزر) في هذه الحالة تُعرض له رساله في الاسفل تخبره بإنه 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 سنتعامل مع مربع النص الذي سنأخذ منه الاسم المطلوب تخزينه، و سنتعامل مع النص الذي نعرض الحاله من خلاله، كذلك نستخدم قواعد البيانات داخل الداله اذاً نحن بحاجه إلى المتغيرين الخاصّين 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 القيمه 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 منّا جملة SQL الخاصه بإضافة البيانات إلى قاعدة البيانات، تكون دائماً دالة execute ضمن المتغير الذي يخزّن المؤشر و في حالتنا هذه المتغير هو cur، اذاً نكتب اولاً السطر الخاص بإضافة البيانات :

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

عندما نقوم بأي تعديل على البيانات، سواء اضفنا او حذفنا او حدّثنا البيانات عن طريق الداله execute، يجب علينا استدعاء الداله commit

حتى نسجّل هذه التغييرات بشكل فعلي في قاعدة البيانات، فإذا قمنا مثلاً بإضافة قيمه جديده في قاعدة البيانات و لم نقم بإستدعاء commit بعدها في حال إغلاق البرنامج تذهب هذه المعلومات التي اضفناها ولا تُسجّل في قاعدة البيانات، و بالتالي انتبه دائماً إلى هذه النقطه حتى لا تضيع المعلومات، تقع الداله دائماً تحت متغير الاتصال و اسم هذا المتغير في حالتنا هو con :

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

الداله 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 الداله 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( '' );
```

انهينا الآن جزء هام في البرنامج، تبقى الجزء الثاني وهو عرض قائمه الاسماء، كما تعلم قمنا بإضافة الاداة TreeView من اجل عرض قائمة الاسماء، في الحقيقه تستخدم هذه الاداة في عملها اسلوب يُسمّى بـ Model-View-Controller و نُطلق عليه MVC اختصاراً، و حتى نتمكن

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 عامّة تواجه المُبرمج، في الآونة الأخير كَثُر إستخدام MVC في تطوير المواقع خصوصاً.

يعتمد هذا الاسلوب على تقسيم العمل إلى ثلاث اجزاء.

الجُزء الاول هو Model و يختص هذا الجزء بالبيانات بمعنى ان البيانات بأنواعها تُخزن 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 الاهم بالنسبه لنا الآن، اذاً يمكننا الاستنتاج انّ الهدف الاساسي من اسلوب MVC هو فصل البيانات عن الطريقة التي تُعرض بها.

نعود للاداة TreeView، كما ذكرنا ان هذه الاداة تستخدم الاسلوب MVC، تُقدّم مكتبة GTK فئه اسمها TreeViewColumn هذه الفئه هي

المسؤوله عن جزء العرض View، كما انها تُقدّم فئتين ListStore و TreeStore و يمكن ا ستخدام احدى هاتين الفئتين كـ Model لتخزين البيانات التي ستُعرض في القائمه، تُستخدم الفئه الاولى من اجل تخزين البيانات التي ستُعرض على شكل قائمه بسيطه، اما الفئه الثانيه فتُستخدم من اجل تخزين البيانات التي ستُعرض على شكل شجره او بالاحرى التي ستُعرض على شكل قائمه تحتوي على هرميات و بيانات تقع تحت بيانات اخرى، و بالطبع لاننا سوف نطبع الا سماء على شكل قائمه بسيطه يجب علينا ا ستخدام الفئه ListStore.

نبدأ اولًا بكتابة جزء العرض (View) مع الاداة TreeViewColumn،
كما ذكرنا ان هناك فئة بنفس الا سم و بالتالي من اجل البدء بجزء العرض نقوم بإنشاء كائن من هذه الا داة و نُكلّف متغير ليكن ا سمه col بها كالتالي :

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

نُلاحظ انّ البارامتر يحتوي على عنوان العمو د الذي يكون في الاعلى، في كُل مره نُنشئ كائن جديد من الصنف TreeViewColumn فإننا بهذه الطريقه نقوم بإنشاء عمو د جديد ضمن قائمتنا، في حالتنا هذه نحن بحاجه

إلى عمودٍ واحدٍ فقط لاننا نريد عرض الاسماء فقط، لو كنّا نريد عرض المزيد من المعلومات مثلاً الاسم و رقم الهاتف و البريد الالكتروني، سنحتاج إنشاء ثلاث كائنات لتُنشئ بدورها ثلاث اعمده.

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

بعدما انشأنا العمود وهو الجزء المُسمى بـ View بإسلوب MVC الذي تستخدمه الاداة TreeView يجب علينا الآن إخبار الأداة 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 بحاجه إلى الجزء View الذي ننشئه من خلال الفئة TreeViewColumn، لنفرض ان القائمه الاولى تحتوي على عمود واحد اسمه "البريد الالكتروني" كذلك بالنسبه 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 انّ الكائن email_col يخص القائمه الاولى ولا يخص القائمه الثانيه؟ حسناً ماذا لو قررنا جعل القائمة 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 كلٌ 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 نُكلّف الكائن col بالقائمه التي يُمثلها الكائن tree كالتالي :

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

هكذا انتهينا من الجزء View، تبقّى جزء Model.

كما اسلفنا انّ الجزء Model هو الخاص بتخزين البيانات، تُقدّم مكتبة PyGTK صنفين احدهما خاص بالقوائم البسيطه العاديه التي استخدمناها في هذا البرنامج (ListStore) و الآخر خاص بالقوائم المُركبه و المُعقده و التي يُطلق عليها اسم الاشجار (TreeStore)، بالطبع سوف نستخدم الصنف ListStore حتى نُنشئ 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 حتماً ستكون نصوص (او string كما نسميها

بالبرمجه). لاحظ الآن البارامتر الذي مررناه عندما نا دينا الصنف ListStore، قمنا بتمرير بارامتر واحد لان لدينا عمود واحد و لان هذا العمود سوف تقع تحته بيانات من نوع النص فهذا البارامتر يوضّح نوع هذه المعلومات عن طريق إرسال اسم النوع، حسناً ماذا لو اضفنا عمودنا الثاني و المعلومات التي تقع هي عمر صاحب الاسم، في هذه الحاله سوف نحتاج لتمرير بارامترين الاول يخص العمود الاول الخاص بالاسماء و بالتالي سوف نرسل str اما الثاني فيخص 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 قُمنا بتكليف الكائن col إلى القائمه التي يمثلها المتغير tree عن طريق السطر :

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

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

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

إنتهينا الآن من إعداد الجزء Model، نُريد الآن تعبئة البيانات التي سوف تُعرض في القائمه، 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 اولًا الداله execute لإرسال الاستعلام، نحتاج إلى اخذ جميع البيانات المُخزنه في جدول names و بالتالي لابد من صياغة إستعلام لتحقيق هذا المطلب بإستخدام لغة SQL :

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

نستقبل الآن ناتج الاستعلام عن طريق الداله fetchall و نُخزنها في names، الناتج عباره عن مصفوفه :

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

اذا كانت هناك معلومات فعلًا فإن حجم names يجب ان يكون اكبر من

0، و بالتالي لابد ان نتحقق ا ذا كان هناك معلومات فعلًا ام قاعدة البيانات فارغه كالتالي :

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

بعدما تأكدنا ان هناك معلومات فعلًا نقوم بقراءة المصفوفه names ووضع محتواها داخل الجزء Model، هذه المصفوفه ثنائية البعد، هناك عدّة مُدخلات و التي تُمثل مُدخلات قواعد البيانات وهو عد د متغير يعو د إلى البيانات التي اضافها المستخدم، و هناك حقلان وهما id و name الموجو دان في الجدول names، ما نريده هو قراءة هذه المُدخلات 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 حلقه تعتمد على حجم المصفوفه names و تقرأ البيانات، و ا ستخدمنا 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 حلّها و لكن يتحسن حلّها، جرّب اضافة اسم جديد من خلال البرنامج، لاحظ انه على الرغم من النجاح في تخزين الاسم بقاعدة البيانات و لكنه لا يظهر في القائمه مبا شره بعد إضافته بنجاح، و لكن يجب علينا إغلاق البرنامج ثم تشغيله مره اخرى حتى يظهر الاسم الجديد ضمن القائمه.

الحل كما خمّنت بسيط جداً، نتوجّه إلى الداله addNewName الخا صه بإضافة الا سماء إلى قاعدة البيانات، نُضيف المتغير store ضمن قائمة المتغيرات التي تقع بعد global حتى يصبح السطر كالتالي :

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

نتوجّه الآن إلى الشرط الذي يتحقق من نجاح اضافة الا سم وهو :

```
if check == None:
    status.set_text( 'تم اضا فة الاسم بـنجاح'
);
    entry.set_text( '' );
```

نستخدم في نهايته الداله append التي تحدثنا عنها منذ قليل لإضافة الا سم إلى القائمه كالتالي :

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

ستصبح الشيفره بهذا الشكل :

```
if check == None:
    status.set_text( 'تم اضا فة الاسم بـنجاح'
```

```
);
    entry.set_text( '' );
    store.append( [ name ] );
```

جرّب الآن و لاحظ ان الاسم يُضاف مباشره إلى القائمه بدون الحاجه إلى إغلاق البرنامج و تشغيله، حسناً و اخيراً انتهينا من هذا المثال الحمدلله :-)

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 شركه ما

لنبدأ الآن ببناء مشروع كامل نستخدم فيه ما تعلمناه، لاحظ انّ هذا الفصل يعتمد بشكل كبير على شرح مثال برنامج الأسماء الموجود في الفصل الثاني و بالتالي تأكد من فهمك الجيّد 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 لتحرير بيانات الموظف، اما الزر الاخير "حذف

الموظف" الذي يعمل بنفس طريقة الزر السابق و لكنه يحذف الموظف بدلًا من تحرير معلوماته.

اربط الحدثين 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 بإسم addNewEmp اما زر التحديث فأربط الحدث clicked مع داله بإسم updateEmp، اما الزر الاخير فأربط الحدث clicked مع داله بإسم deleteEmp، خزّن الملف في مجلد البرنامج و ليكن اسمه gui.glade

نشرع الآن بالشيفره، نبدأ اولًا 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، نشغّل 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();
```

نُنشؤ 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 النافذه الرئيسيه، المتغير tree مسؤول عن القائمه، المتغير add_emp مسؤول عن زر "اضافة 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 القيمه 0 بالنسبه للعمود الاول و كلما زاد عمود نزيد بالرقم، هذا يعني ان العمود الثاني ستكون قيمة البارامتر الثالث الخاص به هي 1 و الثالث يأخذ القيمه 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 );
```

نُضيف 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 قارنّا هذا السطر بالسطر المكتوب بالمثال السابق نلاحظ إننا اضفنا بارامترين و بالتالي ا صبحت لدينا ثلاث بارامترات مُمرره، اما المثال السابق احتوى فقط على بارامتر واحد، لابد و انك تعرف السبب لاننا شرحناه سابقاً :-)، السبب هو انه لدينا ثلاث عواميد هنا و بالتالي كل بارامتر يخص عمود من هذه العواميد و يحدد نوعية البيانات التي تقع تحته، و السطر السابق يُخبر المكتبه بأن العمود الاول يحتوي على نصوص، العمود الثاني يحتوي على اعداد صحيحه و كذلك بالنسبه للعمود الثالث.

نُحدد الآن الكائن الذي تنتمي إليه هذه المعلومات :

```
tree.set_model( 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 القائمه :

```
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 المصفوفه المُمرره إلى الداله append بوجود عواميد متعدده، 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 اولاً دالة اضافة موظف جديد.

عندما يضغط المستخدم على هذا الزر، نُفتح له نافذه جديده تحتوي على ثلاث مربعات نص الاول لكتابة اسم الموظف، الثاني لكتابة راتب الموظف، الاخير لكتابة عمر الموظف، و اسفلهم زر مكتوب عليه "موافق" يضغط عليه المستخدم عندما ينتهي من تعبئة المعلومات حتى تتم اضافة

المعلومات إلى قاعدة البيانات، و اسفل هذا الزر نص الحاله الذي يُكتب فيه "تم اضافة المعلومات بنجاح" عند نجاح اضافة المعلومات، صمم هذه النافذه 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 age، زر موافق نسميه insert_button و اخيراً نص الحاله نسميه status.

لابد من لفت نظرك إلى شئ هام جداً الربط بالحدثين delete-event و destroy هنا غير مهم لان نافذتنا التي نصممها الآن نافذه فرعيه و ليست النافذه الرئيسيه، هذا يعني انها نافذه تظهر عندما نضغط زر محدد في النافذه الرئيسيه، اذا حصل و ربطنا الحدثين في النافذه الفرعيه سوف يؤدي إغلاق النافذه الفرعيه إلى إغلاق البرنامج بالكامل و هذا الذي لا يجب أنْ يحدثَ، إننا بحاجه إلى الربط بحدث واحد فقط وهو clicked الخاص بزر موافق لنربطه بـ Emp.

لنتعلم الآن كيف يمكننا إظهار هذه النافذه و جعلها تؤدي وظيفتها بعد الضغط على زر "اضافة موظف" في النافذه الرئيسيه، نبدأ اولاً بتعريف

: addNewEmp

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

ببساطه لإظهار نافذه فرعيه مُخزنه في ملف منفصل نفعل كما فعلنا مع النافذه الرئيسيه تماماً، نُنشؤ 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' );
```

نُكلّف 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 :-)، نُكمل الآن اخذ الادوات التي سنحتاج

لإستخدامها في الشيفره المصدريه بعد السطر :

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

نأخذ كُل 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 اولاً :

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

add_builder.connect_signals( signals );
```

من الا سطر السابقه تستنتج اننا عندما نضغط على زر الموافقه يتم ا ستدعاء داله بإ سم insertEmp و هي المسؤوله عن إضافة بيانات الموظف إلى قاعدة البيانات، و لكن لما ذا في هذه المره بالذات وضعنا الداله داخل مصفوفه مع مجموعه من الكائنات التي عرفناها مسبقاً؟

قبل كل شئ لنحد د ما الذي نريد الو صول له من الا دوات من داخل الداله insertEmp، سوف نحتاج للو صول إلى مربعات النص الثلاث لإ ستخراج المعلومات منهن و اضافة المعلومات في قاعدة البيانات، بالإضافه إلى ذلك سوف نحتاج إلى نص الحاله لأننا سنغيره في حال نجاح عملية الاضافه او فشلها، جميع هذه الا دوات يمكن الو صول لها من خلال الكائنات التاليه : emp_name لإ سم الموظف، salary لراتبه، age لعمره، status لنص الحاله.

الحل الذي اعتدنا على ا ستخدامه هو ا ستخدام global، هذا يعني اننا لو ار دنا 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! لما ذا 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 الو صول لها بشكل مبا شر ما عدا الداله نفسها، اما المتغيرات العامه فيتم تعريفها خارج جميع الدوال و بالتالي يمكن لجميع الدوال الو صول لها من خلال وضع ا سمها 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 بارمترات إلى الداله التي ننا ديها عند وقوع الحدث مثلما رأيت منذ قليل، مصفوفه نضع فيها اسم الداله اولاً و بعدها البارمترات 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 انّ الداله addNewEmp خاصه بتجهيز و إظهار نافذة الاضافه و الداله insertEmp خاصه بإضافة المعلومات التي تزودها بها الداله الاولى، لنُكمل، بما اننا نريد الوصول إلى الجزء Model في قائمتنا لاننا نريد اضافة الموظف مباشره 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 الأمثَلَه 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 العمليه تمت بنجاح في نص الحاله ثم تفريغ المربعات من المعلومات القديمه حتى تكون اضافة معلومات موظف جديد اسهل مثلما فعلنا مسبقاً، و اخيراً اضافة البيانات مباشره إلى قائمتنا بدلاً من الحاجه إلى إغلاق البرنامج و اعادة فتحه مره اخرى حتى تظهر المعلومات التي اضفناها :

```
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، يحرر المستخدم بيانات الموظف كما يشاء ثم يضغط على زر الموافقه ليتم تحديث المعلومات في قواعد البيانات، يعمل الحذف بنفس الاسلوب تقريباً، 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 تُسمى 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 بإسم 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 بإستخدام الفئة TreeSelection، الفئة الرئيسية TreeView توفر لنا داله اسمها get_selection تُرجع هذه الداله كائن من نوع TreeSelection و يمكننا من خلال هذا الكائن الو صول إلى بيانات الصف المُختار، لنعرّف اولاً الداله updateEmp :

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

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

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

الآن، يعتبر المتغير select كائناً من الفئة TreeSelection، تُقدّم لنا هذه الفئة داله اسمها get_selected و التي تُرجع 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 يُمثّل 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 القيمه 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، حسناً 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 الطريقه 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 البرمجيه :

```
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 تقريباً :

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

edit_builder.connect_signals( signals );
```

لاحظ البارامتر الاول الذي تستقبله الداله editEmp وهو ا سم الموظف المُرا د تحرير بياناته، البارامتر الثاني هو الكائن الذي يتحكم في مربع النص الخاص بإ سم 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، كُل 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 الداله 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 المستخدمَ 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 + '"' );
```

نُغيّر المعلومات بشكل فعلي :

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

نتحقق من نجاح العمليه و نطبع ذلك للمستخدم :

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

هذا كل شئ، لاحظ مدى التشابه بين هذه الداله و بين اختها insertEmp، تكرار الشيفرات بالطبع تُعتبر من العادات البرمجية السيئة، و لو كان هذا البرنامج حقيقياً و لم يكن مجرد مثال تعليمي فمن الأفضل وضع الشيفره المُكرره بين inserEmp و editEmp في داله منفصله ثم إستدعاءها 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 للدالة 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 تماماً مع updateEmp يجب علينا اخذ اسم الموظف الذي نريد حذفه حتى نستخدمه في استعلام الحذف، نعم كما خمّنت تماماً سوف نستخدم الصنف TreeSelection :-)، اولاً 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 الحمدلله رب العالمين و صلواته و سلامه على خير المرسلين و آله الطاهرين، و ما توفيقي 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/
مشروع GNOME : http://www.gnome.org/
مشروع Xfce : http://www.xfce.org/
مشروع GPE : http://gpe.linuxtogo.org/

وصلات دروس

1. دروس 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 بإسم "Using SQLite in Python" على العنوان التالي :
http://www.devshed.com/c/a/Python/Using-SQLite-in-Python

7. دروس بإسم "PyGTK tutorial" على العنوان التالي :
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
3. موقع Glade الرسمي على العنوان التالي : /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