UNIVERSIDAD
DE GRANADA

Estudio de librerías para hacer interfaces gráficas

Periféricos y Dispositivos de Interfaz Humana

Grado en Ingeniería Informática

Curso 2020/21

ALUMNOS
ALONSO BUENO HERRERO
BARTOLOMÉ ZAMBRANA PÉREZ

ÍNDICE DE CONTENIDOS

1 Introducción a las interfaces gráficas

En este trabajo se pretende estudiar las librerías para construir interfaces gráficas para software nativo (no web) más importantes en la actualidad. También se entremezcla este objetivo con el de poder trabajarlas sobre los lenguajes de programación también más conocidos en la actualidad, dentro de las propias limitaciones impuestas por la librería (es decir, para qué lenguajes se ha desarrollado). Una vez aclarados los objetivos que perseguimos alcanzar con este trabajo, presentamos ya las librerías a estudiar:

1. Java **Swing**
2. Tk en Python: **Tkinter**
3. Qt en Python: **PyQT**
4. Wx en Python: **wxPython**

En primer lugar, se escoge la librería Swing de Java por ser todo un clásico en la elaboración de GUI, con un lenguaje del porte de Java, y también porque Swing es la primera (y de las pocas) librería de interfaces gráficas que se nos enseña *por encima* en la carrera (asignatura *Programación y Diseño Orientado a Objetos*, 2.º Curso).

Por otro lado, apostamos por lo más novedoso al trabajar con Python y el desarrollo de GUI con un lenguaje que es “fácil” de nacimiento, pues ya su propio autor lo creó con el objetivo de que fuese así. La selección de las librerías de Python se ha hecho en base a algunos blogs, referenciados al final de este documento.

2 Estudio de librerías

Java Swing

Para entender el origen de Swing es necesario remontarse a los inicios de Java. La primera librería que se definió fue AWT. En su día, esta primitiva librería proporcionaba un conjunto básico de controles, ventanas y cuadros de diálogo que admitían una interfaz gráfica útil, pero limitada. Se hizo evidente que las limitaciones y restricciones presentes en el AWT eran lo suficientemente serias como para necesitar un mejor enfoque. La solución fue Swing. Introducido en 1997, esta librería estaba inicialmente disponible para su uso con Java 1.1 como una biblioteca separada. Sin embargo, a partir de Java 1.2, Swing se integró completamente en Java.

La siguiente tabla resume las diferencias entre Swing y AWT, que son, casi al completo, las grandes ventajas que Swing aportó frente a AWT:

AWT	***Swing***
➔ Depende de la plataforma ➔ No sigue MVC ➔ Componentes menores ➔ No admite *pluggable look and feel*[1] ➔ Heavyweight	➔ independiente de Plataforma ➔ Sigue MVC ➔ Componentes más potentes ➔ Admite *pluggable look and feel* ➔ Lightweight

Los elementos clave: componentes y contenedores

Una interfaz Swing consta de dos elementos clave: ***componentes*** y ***contenedores***, aunque realmente todos los contenedores también son componentes. La diferencia entre los dos se encuentra en su propósito:

- Un componente es un control visual independiente (un botón o campo de texto).
- Un contenedor contiene un grupo de componentes.

Por lo tanto, un contenedor es un tipo especial de componente que está diseñado para contener otros componentes. Además, para que se muestre un **componente**, debe mantenerse **dentro de** un **contenedor**. Esto implica que, en la práctica, todas las GUI de Swing tendrán al menos un contenedor. Como los contenedores son componentes, un contenedor también puede contener otros contenedores. Esto permite a Swing definir lo que se llama una jerarquía de contención, en la parte superior debe ser un contenedor de nivel superior.

En general, los **componentes** Swing se derivan de la clase *JComponent*[2]. *JComponent* proporciona la funcionalidad que es común a todos los componentes, tal y como aparecía en la jerarquía de clases.

En cuanto a los contenedores, Swing define dos tipos:

- contenedores de **nivel superior**: *JFrame*, *JApplet*, *JWindow* y *JDialog*. Heredan las clases de AWT *Component* y *Container*. A diferencia de otros componentes de Swing, que son *lightweight*, los contenedores de nivel superior son ***heavyweight***, un caso especial en la biblioteca de componentes Swing.

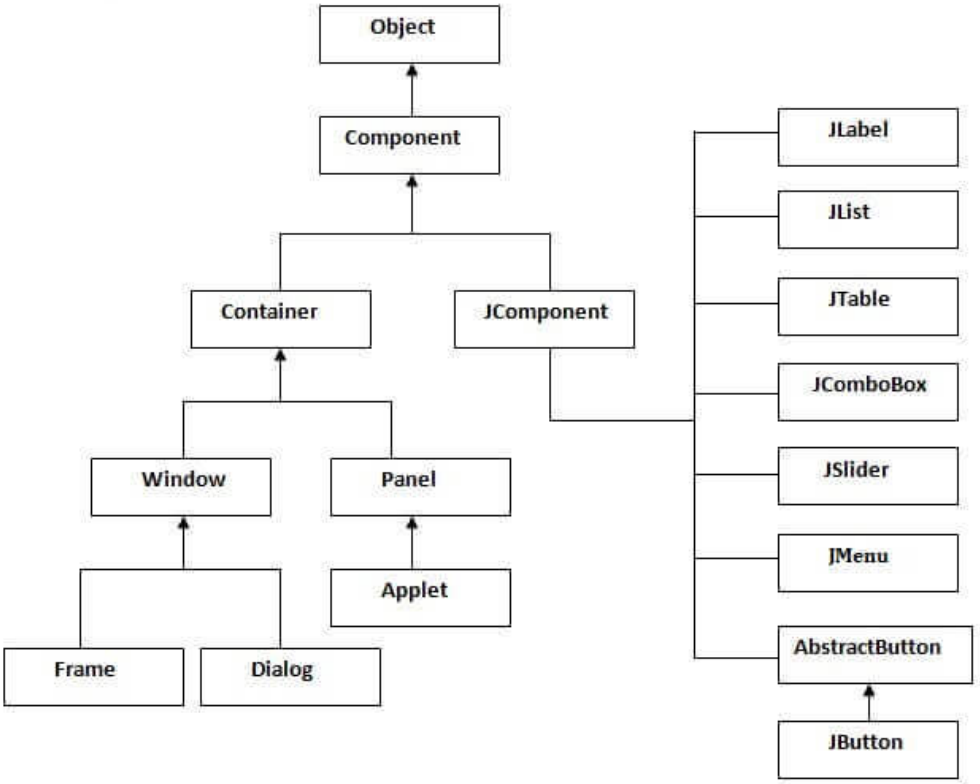
  Como su nombre lo indica, un contenedor de nivel superior debe estar en la parte superior de una jerarquía de contención. Un contenedor de nivel superior no está contenido en ningún otro contenedor. Además, cada jerarquía debe comenzar con un contenedor de nivel superior. El más comúnmente utilizado para las aplicaciones es ***JFrame***. En los ejemplos veremos que lo más común es instanciar JFrame o JApplet. La base de la programación con Swing es programar (“rellenar”) un JFrame o JApplet.

[1] Mecanismo que permite la adaptación de la GUI en tiempo de ejecución. Más información en la entrada que Wikipedia dedica a esta característica en: https://en.wikipedia.org/wiki/Pluggable_look_and_feel.
[2] Las únicas excepciones a esto son los cuatro contenedores de nivel superior, que se describen en la siguiente sección.

- El segundo tipo de contenedor compatible con Swing es el contenedor ***lightweight***. Los contenedores *lightweight* heredan JComponent. Ejemplos de contenedores *lightweight* son JPanel, JScrollPane y JRootPane. Este tipo de contenedores a menudo se usan para organizar y administrar colectivamente grupos de componentes relacionados. Por lo tanto, puede usar contenedores *lightweight* para crear subgrupos de controles relacionados que están contenidos dentro de un contenedor externo.

Jerarquía de clases de la librería

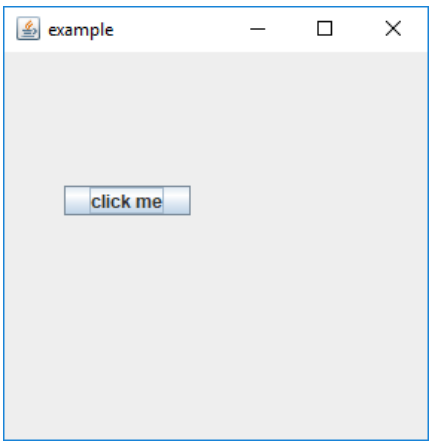
La siguiente figura ilustra la jerarquía de clases de la librería Swing. Se puede apreciar lo que ya comentamos: la clase *Container* es una especialización (un hijo, en términos de herencia OO) de la clase *Component* que alberga a otros objetos tipo *Component*.



Principales componentes

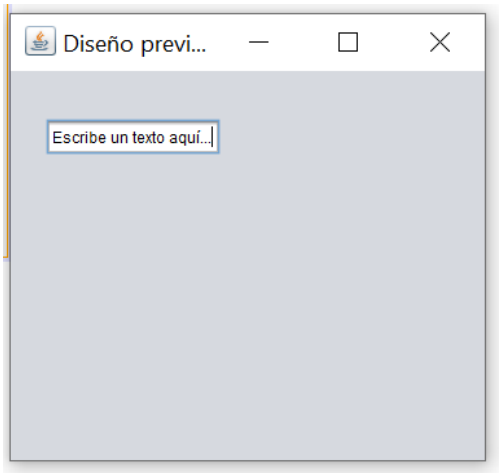
Clase JButton

Se utiliza para crear un botón etiquetado. El uso de *ActionListener* permitirá ejecutar acción cuando se presione el botón. Hereda la clase AbstractButton. Ejemplo:



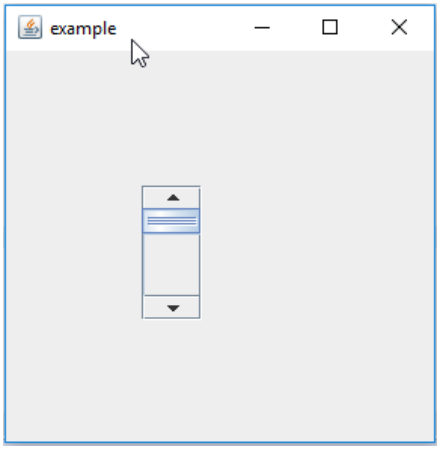
Clase JTextField

Hereda la clase *JTextComponent* y se utiliza para permitir la edición de texto de una sola línea.



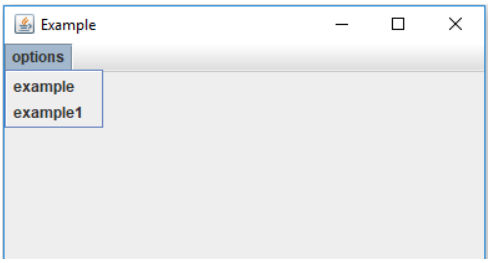
Clase JScrollBar

Se utiliza para agregar una barra de desplazamiento, tanto horizontal como vertical.



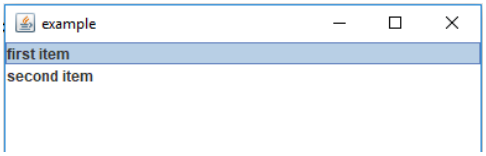
Clase JMenu

Hereda la clase JMenuItem y es un componente de menú desplegable que se muestra en la barra de menú.



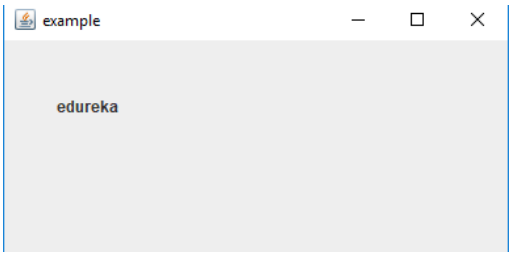
Clase JList

Hereda la clase JComponent, el objeto de la clase JList representa una lista de elementos de texto.



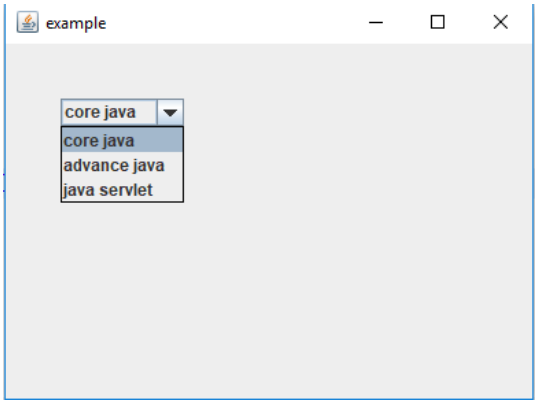
Clase JLabel

Se utiliza para colocar texto en un contenedor. También hereda la clase JComponent.



Clase JComboBox

Hereda la clase JComponent y se usa para mostrar el menú emergente de opciones.



Pasamos ya a ver algunos ejemplos de interfaces gráficas muy básicas usando Swing. Los ejemplos que se presentan son, como cabe esperar, en orden de dificultad creciente, desde el clásico *holamundo* hasta un proyecto algo más complejo, con gestión de ventanas, apertura de archivos, etc.

Ejemplos

Ejemplo 1

Con este ejemplo se pretende ilustrar cómo se crea una ventana con Java Swing y los componentes básicos que tiene.

En primer lugar, cabe recordar que no es necesario que instalemos la librería Swing, ya que las últimas versiones de Java ya lo incluyen de forma predeterminada, por ser la librería básica en Java para hacer GUI.

En segundo lugar, y a modo de comentario previo a la explicación del ejemplo, recomendamos extender la clase JFrame (heredar de ella una clase nueva) en lugar de crear una ventana nueva. Es el mecanismo que emplea el IDE Netbeans y que permite gestionar los eventos de forma mucho más sencilla. Es por ello que el presente ejemplo es meramente ilustrativo de la forma de iniciar una ventana de Swing.

El código necesario es:

```
import javax.swing.*;

public class Principal {

    private static void createAndShowGUI() {

        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel texto = new JLabel("Hola a todos.");
        frame.getContentPane().add(texto);  // añadir texto a la ventana
        frame.pack();                       // necesario
        frame.setVisible(true);          // que la ventana esté visible
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();  // configurador de la GUI
            }
        });
    }
}
```

Explicación de las tareas del código:

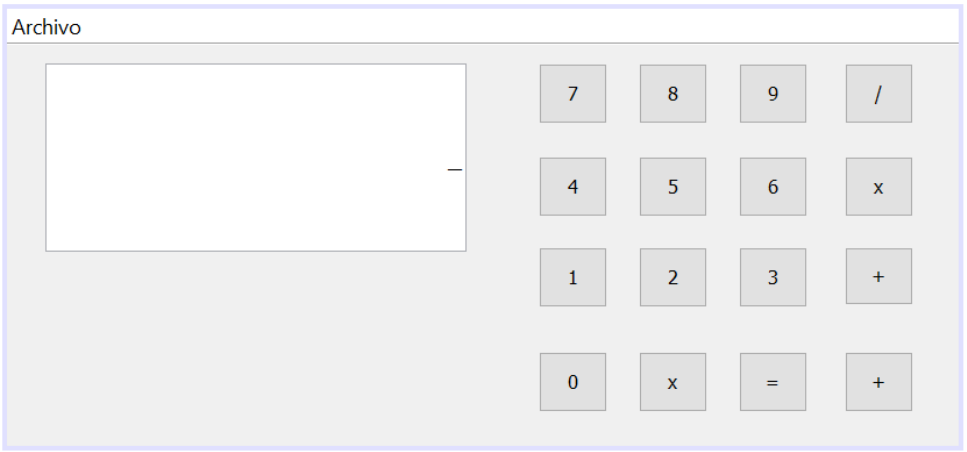
1. Importar la librería.

2. Crear una clase Java (siempre es necesario en Java, como lenguaje Orientado a Objetos)
3. Crear el método *main*. En él usamos el evento *invokeLater*[3], que permite gestionar los eventos de la interfaz en una hebra de proceso aparte del resto de la aplicación para evitar que esta última bloquee la interfaz. Dentro de este método creamos la hebra -objeto *Runnable*()- y lanzamos la hebra mediante el método ***run***(), dentro del cual creamos ya la interfaz llamando a la función ***createAndShowGUI()***
4. La función *createAndShowGUI()* es la encargada de crear el objeto *JFrame* y añadirle un texto aislado (*JLabel*) y mostrar la ventana (*setVisible(true)*).

Ejemplo 2: Uso del gestor de diseño de GUI en Netbeans

Con este ejemplo pretendemos ilustrar cómo usar el gestor gráfico de GUI que proporciona el IDE Netbeans. Este tipo de herramientas son tremendamente útiles, y muy utilizadas, pues nos evitan tener que ir programando línea a línea cada uno de los componentes de una ventana, su posición, tamaño, etc. Con estos gestores, la principal tarea del programador será controlar los eventos e implementar las funcionalidades asociadas a la interfaz, amén de otros detalles de programación, posición, etc mucho más precisos que quiera llevar a cabo.

Diseñaremos una pequeña calculadora que hace las operaciones básicas: suma resta, multiplicación y división, y mostrar los resultados en una pantallita que se limpiará pulsando el botón 'C'. El prototipo es:

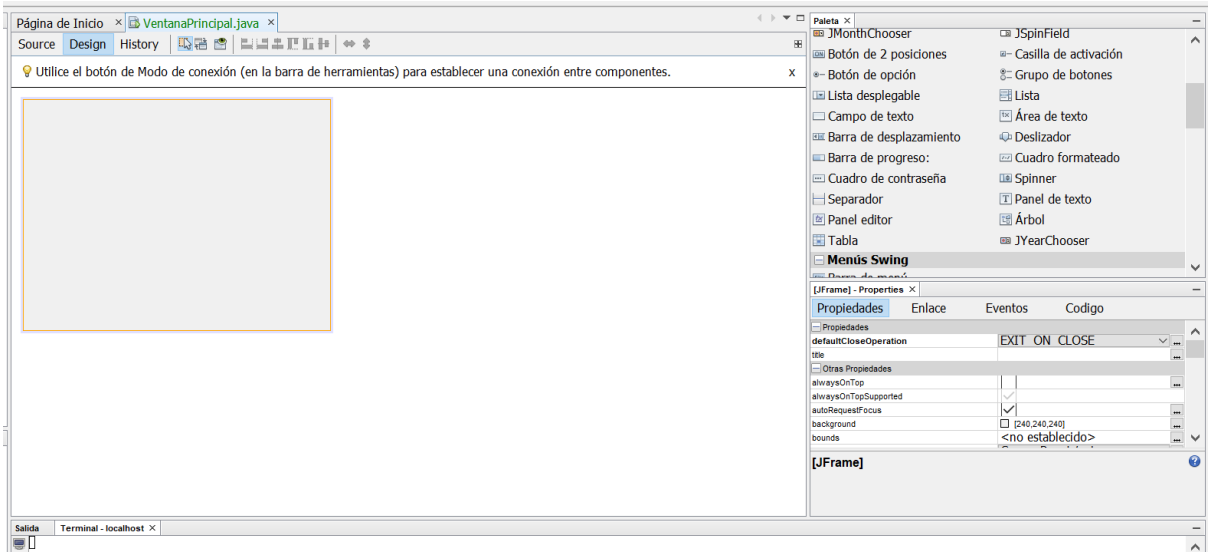


El proceso es el siguiente:

1.- Una vez tenemos instalado el software Netbeans (recomendamos la versión 8.2), iniciamos y creamos un proyecto Java Application.

2.- Pulsamos con el botón derecho del ratón sobre el paquete de fuentes por defecto que se crea en nuestro proyecto (ver barra lateral izquierda de Netbeans), y creamos un nuevo "Formulario JFrame", que es una ventana de Swing. El aspecto es:

[3] Una explicación muy buena en https://elvex.ugr.es/decsai/java/pdf/E4-gui.pdf.



Es decir, se nos abre automáticamente la herramienta de diseño una vez que ha generado el fichero JAVA para la clase JFrame. En esta herramienta ya podemos ir añadiendo elementos diversos a nuestra ventana "pulsando-arrastrando-soltando" dichos elementos hasta el lugar deseado en la ventana. Podemos ver en todo momento el código que se va autogenerando y actualizando cuando estamos en modo "Diseño" si cambiamos al modo "Source", en la barra de herramientas superior al espacio de trabajo blanco.

3.- Para gestionar el evento asociado a un botón, basta con pulsar "doble click" sobre él, y nos llevará a la vista de código, a una función, inicialmente vacía, que se ejecutará cada vez que, durante la ejecución de la aplicación, se pulse dicho botón; es un objeto "actionPerformed", que, en efecto, agrupa las tareas asociadas al evento ***pulsar el botón***. Tienen la forma:

```
private void botonCalcularActionPerformed(ActionEvent evt) {
    // código a ejecutar cuando se pulse botonCalcular
    System.out.println("Has pulsado el botón de cálculo.");
}
```

4.- También se puede insertar un objeto *JTextField* que permite gestionar una caja donde se puede tanto leer texto como escribirlo desde el código para que el usuario de la GUI lo vea. Es el elemento que se usa para el prototipo de la calculadora que hemos presentado.

5.- Para ejecutar la interfaz, el IDE ha generado también una función main dentro del código de la clase, con el mismo esquema que explicábamos en el Ejemplo 1, aunque si vamos a tener varias ventanas en el proyecto, habrá que saber dónde estará el main, si usamos un patrón de diseño, etc.

Tk en Python: *Tkinter*

Introducción

Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows.

Hoy en día hay otras alternativas disponibles como wxPython o PyQt, que veremos a más adelante, y que cumplen con todos los estándares de componente visual.

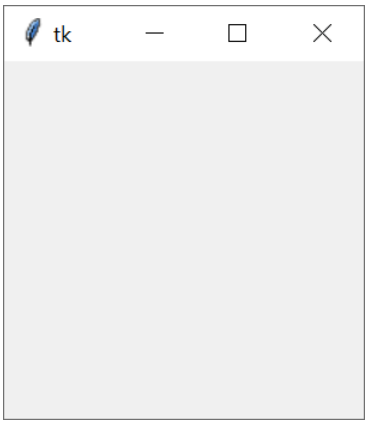
¿Cómo obtener esta librería de Python?

Una vez hemos instalado Python (software libre, gratis), tendremos instalada la librería Tkinter, pues, como ya se ha comentado, es la ídem base en este lenguaje para la elaboración de GUI.

Widgets

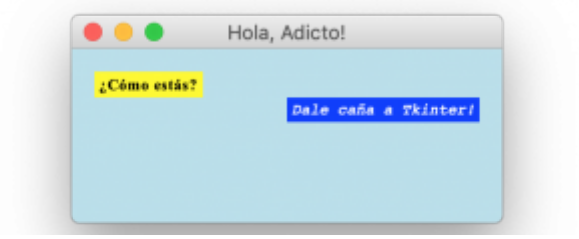
Si en Swing teníamos *componentes* y *contenedores*, aquí tenemos los denominados ***widgets***. Los widgets son los componentes que vamos a añadir a nuestra interfaz. A la hora de montar una GUI con Tkinter, nos basaremos en widgets *jerarquizados*, que la irán componiendo poco a poco. Algunos de los más comunes son:

- ***Tk***: es la raíz de la interfaz, donde vamos a colocar el resto de widgets.

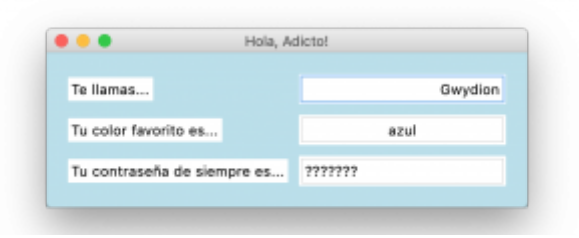


- *Frame*: marco que permite agrupar diferentes widgets.

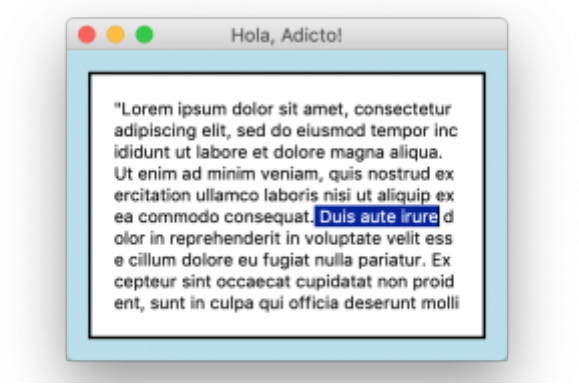
- *Label*: etiqueta estática que permite mostrar texto o imagen.



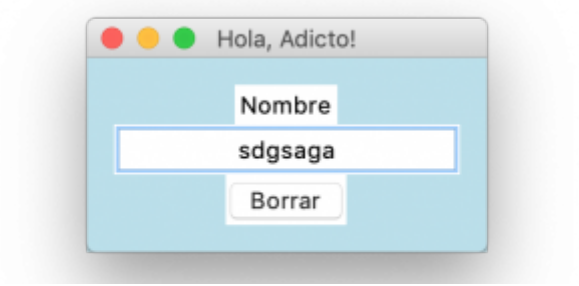
- *Entry*: etiqueta que permite introducir texto corto (típico de formularios).



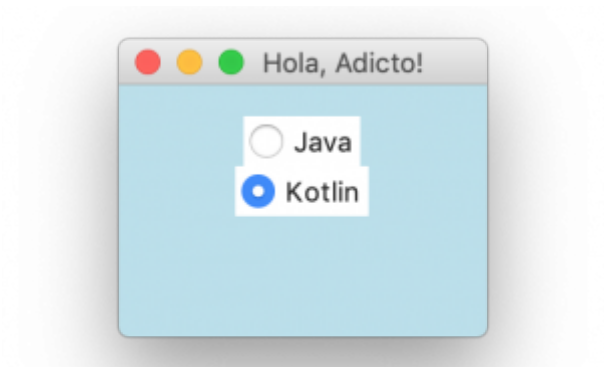
- *Text*: campo que permite introducir texto largo (típico para añadir comentarios).



- *Button*: ejecuta una función al ser pulsado.

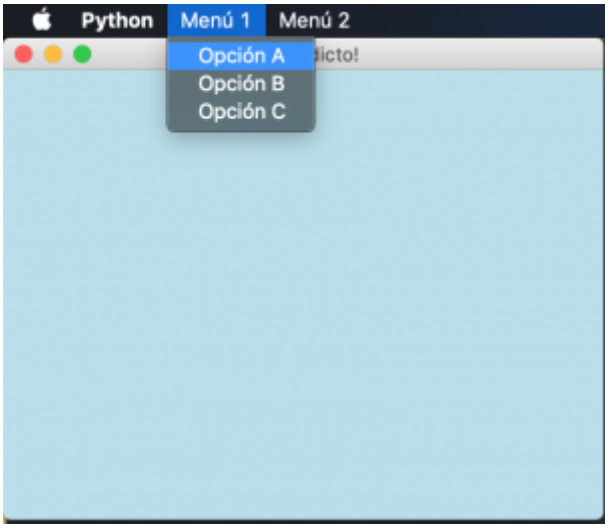


- *Radiobutton*: permite elegir una opción entre varias.



- *Checkbutton*: permite elegir varias de las opciones propuestas.

- *Menu*: clásico menú superior con opciones (Archivo, Editar…).



- Dialogs: ventana emergente (o pop-up).

Nota: las imágenes anteriores se han obtenido de:
https://www.adictosaltrabajo.com/2020/06/30/interfaces-graficas-en-python-con-tkinter/.

Cuando vayamos a inicializar el componente, debemos pasar por constructor el elemento que quede "por encima" en la jerarquía de la vista (si queremos colocar una label dentro de un frame, al construir la etiqueta le pasaremos el marco como argumento del constructor).

Ejemplos

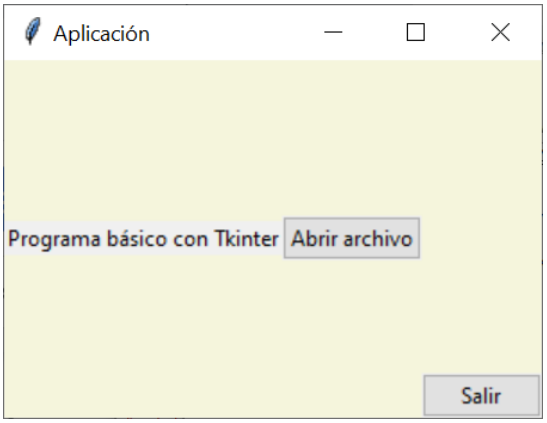
Ejemplo 1

Vamos a mostrar un ejemplo básico de un programa que implementa una típica funcionalidad "Open" (seleccionar archivo de nuestro PC) y que muestra el nombre completo del fichero seleccionado en una nueva ventana. Presta atención a los comentarios a lo largo del código.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk  # widgets temáticos (módulo tkinter.ttk)
from tkinter.filedialog import askopenfilename

# Crea una clase Python para definir el interfaz de usuario de
# la aplicación. Cuando se cree un objeto del tipo 'Aplicacion'
# se ejecutará automáticamente el método __init__() qué
# construye y muestra la ventana con todos sus widgets:

class Aplicacion():
    def __init__(self):
        raiz = Tk()                         # crear la ventana
        raiz.geometry('300x200')            # definir tamaño en pixeles
        raiz.configure(bg = 'beige')        # definir fondo de la ventana
        raiz.title('Aplicación')            # título de la ventana

        # añadir componentes a la ventana: texto y 2 botones
        ttk.Label(raiz, text="GUI básica con Tkinter").pack(side=LEFT)
        ttk.Button(raiz, text='Abrir archivo',
                   command=Aplicacion.abrir_archivo).pack(side=LEFT)
        ttk.Button(raiz, text='Salir',
                   command=raiz.destroy).pack(side=BOTTOM)

        raiz.mainloop()      # bucle infinito de ejecución de la GUI
```

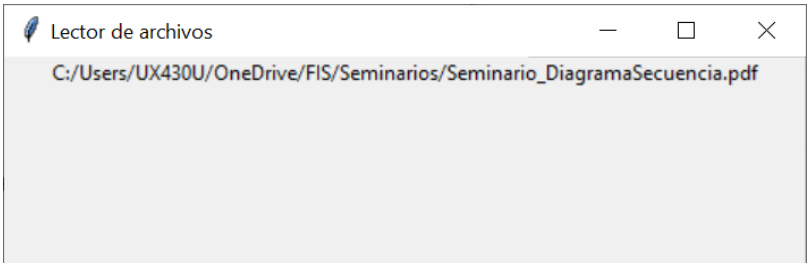
```
    def abrir_archivo():
        filename = askopenfilename()
        print(filename)
        v1 = Tk()
        v1.geometry('300x200')
        ttk.Label(v1, text=filename).pack()


# Definir la función main(): que indica
# el comienzo del programa. Dentro de ella se crea el objeto
# aplicación 'mi_app' basado en la clase 'Aplicación':

def main():
    mi_app = Aplicacion()  # instanciamos la clase Aplicacion
    return 0


# __name__ es una variable especial de python que constituye el main del
programa,
# la semilla de ejecución, por decirlo así
if __name__ == '__main__':
    main()
```

El resultado es el siguiente:



Y si pulsamos…

- el botón “Abrir archivo” podremos seleccionar un archivo de nuestro PC. Al darle a aceptar, se abrirá una nueva ventana que muestre el nombre del fichero seleccionado, como se muestra en la imagen siguiente:



- el botón “Salir” el programa finalizará con código “0”, indicando que lo ha hecho sin errores.

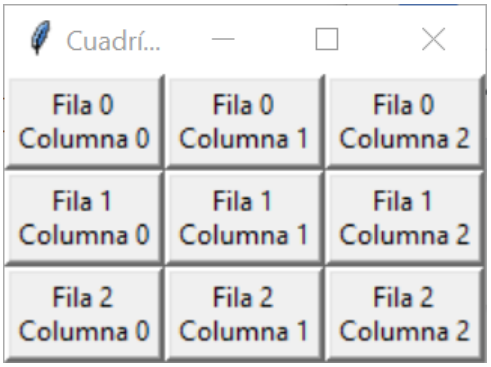
Ejemplo 2: Cuadrícula de botones con elementos Frame

Ejemplo adaptado del propuesto en www.realpython.com.

Este es un curioso ejemplo que hemos encontrado en uno de los portales sobre Python más completos de la red: https://realpython.com/. Construye una cuadrícula de botones usando el componente .grid() de los frames, y botones dentro de cada casilla de esa cuadrícula. El código, bastante simple, es el siguiente:

```
import tkinter as tk

window = tk.Tk()
window.title("Cuadrícula")

for i in range(3):  # por filas
    for j in range(3):  # por columnas

        frame = tk.Frame(
            master=window,     # frame dentro de la ventana "window"
            relief=tk.RAISED, # sin espacio entre cuadros => más estético
            borderwidth=1
        )
        frame.grid(row=i, column=j)
        boton = tk.Button(master=frame,
                          text=f"Fila {i}\nColumna {j}").pack()

window.mainloop()    # bucle principal
```

Y el resultado que se produce es el siguiente:



¿Qué tareas hemos realizado en el código?

1. Importar la librería tkinter.
2. Creamos una ventana y le asignamos un título.
3. Dentro de un doble bucle vamos a construir la cuadrícula de botones:
    a. En cada iteración del “doble bucle”, es decir, dentro del *for* más interno, creamos un objeto frame;

   b. con la propiedad .grid() le indicamos que estará ubicado en la fila ***i*** y en la columna ***j*** (argumentos de .*grid*());
   c. dentro de ese *frame*, creamos un botón como ya lo hicimos en el ejemplo anterior, y le asignamos como texto del botón la fila y columna en que está. Adicionalmente, se le podría asignar una función que actuara al pulsar el botón y otras propiedades.
4. Este doble bucle generará, tal y como indica el código, una cuadrícula de 3x3 con cada cuadro rellenado con un botón.
5. Llamamos al bucle mainloop(), como siempre.

PyQT5

Introducción

En el año 1999 apareció PyQt, un binding de Qt creado por la compañía inglesa Riverbank Computing, distribuido bajo varias licencias diferentes.

Se trata de un conjunto de bibliotecas C + + multiplataforma que implementan una API de alta nivel para acceder a muchos aspectos de los asustemos actuales.

Se puede decir que es un conjunto completo de enlaces de *QT5 for Python.* Dado el hecho de que se trata de un conjunto de bibliotecas C + + multiplataforma implica que se pueda integrarse PyQt5 con aplicaciones C + +.

Por lo tanto, PyQt5 es compatible con plataformas Windows, Linux, UNIX, Android, macOS e IOS requiriendo Python3.5 o superior.

Licencia

PyQT5 se publicó con licencia GPL v3 y bajo licencia comercial la cual permite el desarrollo de aplicaciones propietarias. Cabe destacar que la licencia ha de ser compatible con la licencia de QT, dado al hecho que hace uso del código de Qt pero a diferencia no se encuentra disponible bajo la licencia LGPL.

Componentes

Todos los módulos de extensión se encuentran instalados en el paquete de Python *PyQT5,* siendo algunos los siguientes:

- Enginio
- Qt
- Qt3DAnimation
- Qt3DCore
- QtCore

> *El resto de módulos los podemos encontrar en:*
> *https://www.riverbankcomputing.com/static/Docs/PyQt5/module_index.html#ref-module-index*

PyQt5 contiene complementos que permiten el desarrollo de nuestra interfaz desde Qt Designer, facilitando mucho la creación de la misma. De ahí que cuente con PyQt5 con el programa *pyuic5*, permitiendo convertir las GUI basados en QtWidgets creadas con QT Designer en código Python.

Conceptos básicos

Los conceptos básicos a dominar para la realización de interfaces de forma eficiente son los siguientes:

- Aplicaciones.
- Bucle de eventos.
- Ventanas
- Widgets.
- Administradores de diseño (Medidores).
- Diálogos.
- Eventos y Bucle de Eventos.
- Señales y Ranuras.

Estos elementos o componentes básicos se encuentran en prácticamente todas las aplicaciones GUI. La mayoría de ellas están representadas como clases dentro del paquete principal *QtWidgets.*

Tratemos cada uno de ellos a continuación.

Aplicaciones

Cualquier aplicación que queramos desarrollar ha de tener una instancia del objeto *QAplication* para gestionar el flujo de la aplicación (iniciación, finalización …)

Ventanas principales

Nos posibilitan la creación de barras de menú, barras de herramientas y de estado. Para ello, hemos de crear una clase que herede de QMainWindow, de modo que cualquier instancia que derive de QMainWindow se considera una ventana principal.

Por defecto se tiene un diseño integrado que se puede utilizar siendo el siguiente:

- Varias barras de herramientas que se encuentra al lado de la ventana.
- Un menú en la parte superior.
- Un widget central que se encuentra dentro de la ventana
- Una barra de estado que se encuentra en la parte inferior de la ventana.

Cabe destacar que para poder utilizarla se ha de tener un widget central definido.

Posteriormente se establecerá un ejemplo de implementación.

Widgets

Qwidgets es la base para todos los objetos de la interfaz de usuario. Todos los componentes definidos sobre este paquete poseen una forma rectangular que se pueden colocar en la ventana para construir la GUI.

Cada widget que se encuentra en el paquete que nos encontramos describiendo son capaces de registrar las pulsaciones, movimientos del ratón entre muchos otros eventos, por lo que podemos anunciar y realizar metodologías de cambio de estados.

Los más utilizados son los siguientes:

- *QPushBotton*: se trata de boton.
- *QLabel*: etiquetas para mostrar información útil, se puede establecer en código HTML.
- *QLineEdit*: consiste en un cuadro de texto de una sola línea en la que se necesita que el usuario introduzca texto.
- *QComboBox*: se trata de un cuadro combinado que muestra una lista de opciones al usuario.
- *QRadioButton*: se trata de un selector de opciones.

Administración de diseño (Medidores)

Consiste en un conjunto de clases que nos posibilitan el ajuste del tamaño y colocar los widgets en las posiciones deseadas de la interfaz. También nos posibilita ajustar el tamaño de los widgets en la ventana.

Estas clases son:

- QHBoxLayout: Organiza los widgets horizontalmente.
- QVBoxLayout: Organiza los widgets verticalmente.
- QGridLayout: Organiza los widgets en una cuadrícula de filas y columnas.
- QFormLayout: Organiza los widgets en filas de dos columnas.

Posteriormente, establecemos en el apartado ejemplos, un ejemplo de cada uno.

Diálogos

Cabe destacar el hecho de que en el caso de implementar una interfaz de diálogos hemos de crear una una clase GUI que herede de *QDialog*. También hay que decir que un diálogo siempre consiste en un widget de nivel superior.

Eventos - Bucle de eventos, Señales y Ranuras.

Cualquier aplicación se encuentra controlada por eventos, como puede ser un click en un botón, o una tecla del teclado. Para manejar los eventos lo que se realizar es establecer un bucle de eventos. Para cada iteración se verifica que no ha ocurrido un evento *terminate*.

El ciclo de eventos se inicia y finaliza mediante el objeto *QAplication* llamando al método *exec_*().

Encontramos widgets como un botón que es capaz de recibir una señal de pulsación, pero la señal por si misma no realiza ninguna acción por lo que hay que asignarle una *ranura*. Es decir, una función o método que se encarga de la ejecución de una acción asociada a dicha señal.

Podemos conectar varias señales en serie y del mismo modo una señal con varias ranuras. Método de implementación:

```
<widget><type_of_signal>.connect(<function>)
```

Instalación de la versión GPL

Para instalar esta PyQt5 únicamente tenemos que realizar el comando

```
pip install PyQt5
```

o también

```
pip3 install PyQt5
```

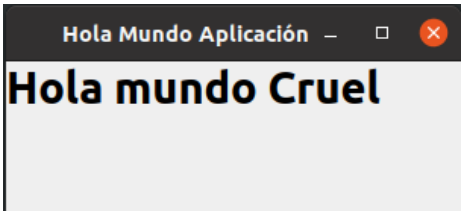
Ejemplos

Ejemplo 1. Hola mundo.

```
#Importamos system para permitir manejar el estado de salida.
import sys

#Importamos las clases necesarias.
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QLabel
from PyQt5.QtWidgets import QWidget

# Creamos un objeto QApplication para crear la aplicación, toda
# aplicación QtPy5 ha de crearlo. Proporcionándonos información global.
# Se le pasa sys.argv ya que utiliza el constructor para crear la
aplicación
# utilizando argc y argv en c++.
app = QApplication(sys.argv)


# Creamos la ventana de la aplicación. es decir la GUI de la aplicación.
# Para ello nos bastamos con un QWidget.
window = QWidget()

# Establecemos el título de la ventana
window.setWindowTitle('Hola Mundo Aplicación')

# Establecemos el tamaño de la ventana.
# Primer y segundo argumento xy, tercer y cuarto anchura y altura
# window.setGeometry(100,300,300,100)
```

```
# Establecemos la posición de la ventana.
window.move(60,15)

# Para mostrar texto en la GUI hay que establecer un objeto QLabel.
# Los objetos pueden aceptar HTML lo cual puede llegar a ser muy funcional.
# A la hora de crearlo hay que asociar la ventana en la que queremos que se
# muestre. Cabe destacar que se puede utilizar cualquier widget
# como ventana de nivel superior (botón, etiqueta)
mensaje = QLabel('<h1>Hola mundo Cruel</h1>',parent=window)



#Mostramos la venta.
window.show()

# Hacemos que se ejecute en bucle la interfaz
# se establece dentro de sys.exit para limpiar recursos.
sys.exit(app.exec_())
```

Resultado:

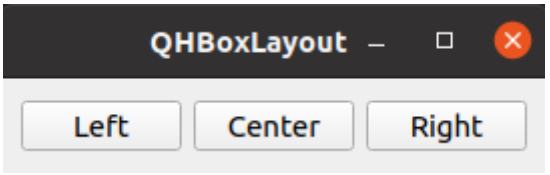


Ejemplo 2: QHBoxLayout

```
#Importamos system para permitir manejar el estado de salida.
import sys


#Importamos las clases necesarias.
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QHBoxLayout
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QWidget

# Creamos un objeto QApplication para crear la aplicación, toda
# aplicación QtPy5 ha de crearlo. Proporcionándonos información global.
# Se le pasa sys.argv ya que utiliza el constructor para crear la
# aplicación
# utilizando argc y argv en c++.
app = QApplication(sys.argv)

# Creamos la ventana de la aplicación. es decir la GUI de la aplicación.
# Para ello nos bastamos con un QWidget.
window = QWidget()
window.setWindowTitle('QHBoxLayout')

#Establecemos el layout para la adición de widgets, en este caso botones.
```

```
layout = QHBoxLayout()
layout.addWidget(QPushButton('Left'))
layout.addWidget(QPushButton('Center'))
layout.addWidget(QPushButton('Right'))

#Añadimos el layout a la ventana
window.setLayout(layout)
#Mostramos la ventana.
window.show()
#Establecemos el bucle de eventos.
sys.exit(app.exec_())
```

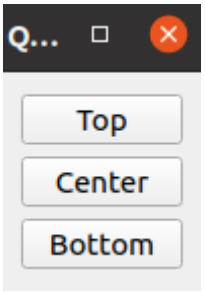
Resultado:



Ejemplo 3 QVBoxLayout

```
#Importamos system para permitir manejar el estado de salida.
import sys

#Importamos las clases necesarias.
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QVBoxLayout
from PyQt5.QtWidgets import QWidget

# Creamos un objeto QApplication para crear la aplicación, toda
# aplicación QtPy5 ha de crearlo. Proporcionándonos información global.
# Se le pasa sys.argv ya que utiliza el constructor para crear la
aplicación
# utilizando argc y argv en c++.
app = QApplication(sys.argv)

# Creamos la ventana de la aplicación. es decir la GUI de la aplicación.
# Para ello nos bastamos con un QWidget.
window = QWidget()
window.setWindowTitle('QVBoxLayout')

#Establecemos el layout para la adición de widgets, en este caso botones.
layout = QVBoxLayout()
layout.addWidget(QPushButton('Top'))
layout.addWidget(QPushButton('Center'))
layout.addWidget(QPushButton('Bottom'))

#Añadimos el layout a la ventana
window.setLayout(layout)
#Mostramos la ventana.
window.show()
#Establecemos el bucle de eventos.
sys.exit(app.exec_())
```

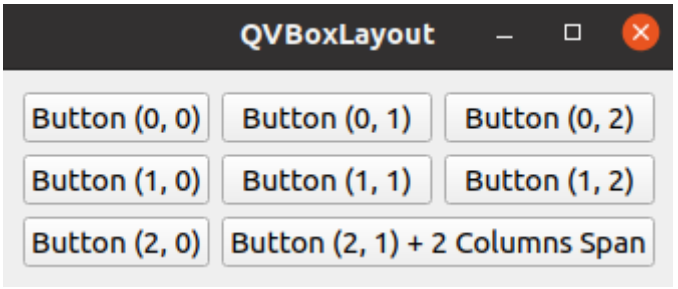
Resultado:



Ejemplo 4 QGridLayout

```
#Importamos system para permitir manejar el estado de salida.
import sys

#Importamos las clases necesarias.
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QGridLayout
from PyQt5.QtWidgets import QWidget

# Creamos un objeto QApplication para crear la aplicación, toda
# aplicación QtPy5 ha de crearlo. Proporcionándonos información global.
# Se le pasa sys.argv ya que utiliza el constructor para crear la
aplicación
# utilizando argc y argv en c++.
app = QApplication(sys.argv)

# Creamos la ventana de la aplicación. es decir la GUI de la aplicación.
# Para ello nos bastamos con un QWidget.
window = QWidget()
window.setWindowTitle('QVBoxLayout')

#Establecemos el layout para la adición de widgets, en este caso botones.
layout = QGridLayout()
layout.addWidget(QPushButton('Button (0, 0)'), 0, 0)
layout.addWidget(QPushButton('Button (0, 1)'), 0, 1)
layout.addWidget(QPushButton('Button (0, 2)'), 0, 2)
layout.addWidget(QPushButton('Button (1, 0)'), 1, 0)
layout.addWidget(QPushButton('Button (1, 1)'), 1, 1)
layout.addWidget(QPushButton('Button (1, 2)'), 1, 2)
layout.addWidget(QPushButton('Button (2, 0)'), 2, 0)
layout.addWidget(QPushButton('Button (2, 1) + 2 Columns Span'), 2, 1, 1, 2)

#Añadimos el layout a la ventana°
window.setLayout(layout)
#Mostramos la ventana.
window.show()
#Establecemos el bucle de eventos.
sys.exit(app.exec_())
```

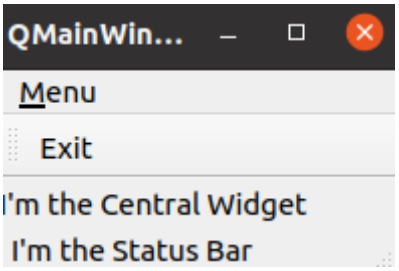
Resultado:



Ejemplo 5 Ventana

```
import sys

from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QLabel
from PyQt5.QtWidgets import QMainWindow
from PyQt5.QtWidgets import QStatusBar
from PyQt5.QtWidgets import QToolBar

class Window(QMainWindow):

    def __init__(self, parent=None):

        super().__init__(parent)
        self.setWindowTitle('QMainWindow')
        self.setCentralWidget(QLabel("I'm the Central Widget"))
        self._createMenu()
        self._createToolBar()
        self._createStatusBar()

    def _createMenu(self):
        self.menu = self.menuBar().addMenu("&Menu")
        self.menu.addAction('&Exit', self.close)

    def _createToolBar(self):
        tools = QToolBar()
        self.addToolBar(tools)
        tools.addAction('Exit', self.close)

    def _createStatusBar(self):
        status = QStatusBar()
        status.showMessage("I'm the Status Bar")
        self.setStatusBar(status)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec_())
```

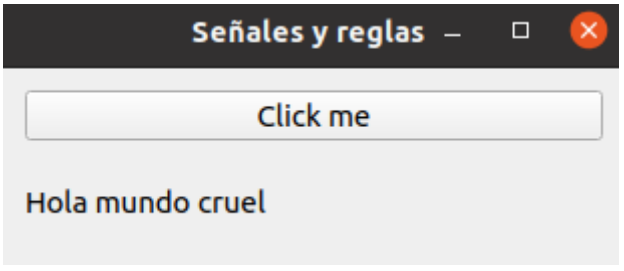
Resultado:



Ejemplo 6. Eventos

```
import sys

from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QLabel
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QVBoxLayout
from PyQt5.QtWidgets import QWidget

def greeting():
   if msg.text():
       msg.setText("")
   else:
       msg.setText("Hola mundo cruel")

app = QApplication(sys.argv)
window = QWidget()
window.setWindowTitle('Señales y reglas')
layout = QVBoxLayout()

btn = QPushButton('Click me')
btn.clicked.connect(greeting)  # Connect clicked to greeting()

layout.addWidget(btn)
msg = QLabel('')
layout.addWidget(msg)
window.setLayout(layout)
window.show()
sys.exit(app.exec_())
```

Resultado:



La librería *wxPython*

Introducción

Surge justo antes del cambio de siglo, con la necesidad de unión de la biblioteca *wxWidgets* con *Python* para permitir la implementación de una GUI compatible con sistemas HP-UX y Windows 3.1. Aunque cabe destacar que al principio era una pesadilla su utilización, esto se debe a consecuencia de que todo código o módulo fue elaborado a mano y pequeños cambios o mejoras de la biblioteca *wxWidgets* significaba cambios o errores importantes en distintos lugares.

Esta librería significó, para su desarrollador, Robin Dunn, el paso de un simple proyecto personal a un proyecto de desarrollo en equipo, con diseños y fechas límites.

Fue alrededor de 2012 cuando *wxPython* empieza a coger fuerza con el conocido proyecto ***Phoenix***, un proyecto en el que se realiza el esfuerzo de compatibilizar wxPython con Python 3.

Con el fin de mejorar los problemas de velocidad y mantenimiento del primer “prototipo” se realiza una nueva implementación envolviendo la biblioteca wxWidgets. Por lo tanto, permite a las aplicaciones Python tener interfaces de usuario en sistemas Windows, Mac y Unix con un aspecto y comportamiento nativo, requiriendo escaso código de cada plataforma específica.

Algunos casos de éxito de esta librería:

- Trastana: herramientas sofisticadas para el análisis cualitativo de datos de texto, imágenes fijas, audio y video.

- Vintech RCAM-Pro: sistema CAM, especialmente diseñado para el anidamiento y programación de formas reales de máquinas CNC para corte térmico y por chorro de piezas de chapa.

- GNUMed: registro médico electrónico (EMR) gratuito / libre para sistemas similares a Unix (sistemas BSD, Linux y UNIX), Microsoft Windows, macOS y otras plataformas. GNUmed tiene como objetivo proporcionar software médico que respete la privacidad de los pacientes y que se base en estándares abiertos.

- Robot Framework IDE: editor de datos de prueba para datos de prueba de Robot Framework. Navegue por las carpetas de Test Suites y ejecútelas filtrando por Etiquetas.

- OutWiker: diseñado para almacenar notas en un árbol. Estos programas se denominan "outliner", wiki personal o editores en forma de árbol. La principal diferencia de OutWiker con otros programas similares es mantener el árbol de notas en forma de directorios en el disco y fomentar el cambio de base por fuentes y programas externos.

- BetaMatch: sistema de software muy especializado para la combinación de componentes de antenas.

- DisplayCAL: solución de perfilado y calibración de pantallas que se centra en la precisión y versatilidad.

- Pyspread: aplicación de hoja de cálculo no tradicional que se basa y está escrita en el lenguaje de programación Python

- Alerce: biblioteca y un juego de herramientas de código abierto para procesar y analizar datos científicos.

- Dicompyler: plataforma extensible de investigación de radioterapia de código abierto basada en el estándar DICOM.

Licencia

La licencia que posee es LGPL al igual que *wxWidgets* ya que hace uso de la misma.Aplicaciones en las que se ha empleado

Conceptos

Para desarrollar una interfaz de forma eficiente hemos de tratar los conceptos más básicos como son los siguientes:

- Bucles de eventos.
- Widgets
- Posicionamiento Absoluto.
- Medidores (dimensionamiento dinámico)
- Eventos.
- Aplicación.
- Ventana de la aplicación (Frame).

Bucles de eventos

Cuando el usuario realiza una acción sobre la interfaz se considera un evento. Cabe decir que el kit de herramientas utiliza un bucle infinito a lo que se le denomina **bucle de eventos.**

El bucle de eventos solo espera a que ocurran los eventos que han sido desarrollados por los usuarios. Por lo tanto han de desarrollarse controladores para su detección.

Para poner en funcionamiento un bucle infinito, ha de declararse un tipo de objeto *wx.App.* estableciendo al final de la codificación el llamado a su método MainLoop().

Widgets

wxPython cuenta con una amplia variedad de widgets posibilitando la construcción rica de interfaces de usuarios. Widgets básicos, así como interesantes de tratar.

- *wx.Panel*: Es un widget no obligatorio pero si recomendable, ya que posibilita el correcto visualizado para la plataforma window. Se puede decir que el panel es la ventana principal, es decir lugar donde se encuentra el cuerpo de la ventana principal.
- *wx.TextCtr:* Se trata de un cuadro de texto que posibilita la introducción del mismo por el usuario.
- *wx.Button:* Se trata de un botón al cual se le puede asociar un evento, así como su posicionamiento absoluto.

Posicionamiento Absoluto

Se denomina posicionamiento absoluto cuando se le proporciona las coordenadas exactas de su posicionamiento a un widget.

Cabe destacar que cuando la aplicación se vuelva más compleja con un mayor número de elementos puede llegar a ser difícil su mantenimiento, Por lo tanto, se recomiendo la utilización de medidores.

Medidores

Los medidores nos permiten la realización de diseños dinámicos certificándonos que la ubicación de los widgets se ajustarán por el tamaño de la ventana.

Los medidores principales son:

- *wx.BoxSizer:* organiza los widgets en una caja donde se pueden agregar los elementos.
- *wx.GridSizer:* organiza los widgets en una cuadrícula de filas y columnas.
- *wx. FlexGridSizer:* nos permite organizar medidores dentro de un propio medidor.

Eventos

Nos permiten la realización de acciones tras la detección de algún evento producido por el usuario, como por ejemplo puede ser la pulsación sobre un botón.

Para asociar un evento a un widget se realiza mediante la llamada al método *Blind*, pasando como primer parámetro el evento y como segundo la función a ejecutar tras la detección del evento.

```
<widget>.Blind(<evento>,<función>)
```

Por convenio la función que se le pasa por argumento ha de tener dos argumentos:

- self, para pasar la instancia del objeto que definimos por si queremos ejecutar algún método.
- event: evento.

Aplicación

Para la realización de cualquier aplicación se ha de declarar un objeto *wx.App()*. Este objeto nos posibilitará la llamada al bucle de eventos como mencioné anteriormente.

Ventana de la aplicación (Frame)

wx.Frame se encarga de la creación de una ventana para que el usuario interactúe. Puede darse caso que un frame tenga otro frame padre asociado.

Cabe decir que de forma predeterminada cuenta con botones para la minimización, maximización para para salir de la aplicación. Sin embargo, para disfrutar de todas las funcionalidades pudiendo añadir widgets al frame se realiza una clase que hereda de la que nos encontramos describiendo.

Todo esto se puede observar en los ejemplos descritos al final.

Para ampliar

Para ver más ejemplos, de diferentes niveles de complejidad, recomendamos https://zetcode.com/wxpython/.

Instalación

Para su instalación basta con realizar:

```
pip install  wxpython
```

El tiempo de instalación suele ser elevado.

También es posible instar la librería con el gestor *pip3*:

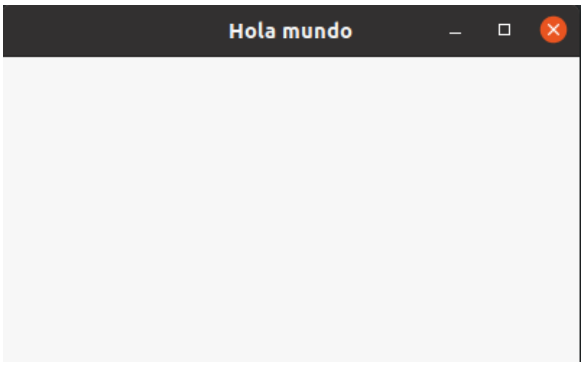
```
pip3 install  wxpython
```

Ejemplos

Ejemplo 1

```
#Primera cosa que hemos de realizar. Importar el
paquete.
import wx;

#Creamos el objeto de aplicación que se encontrará
constantemente ejecutándose.
app = wx.App()

#Creamos el frame, lo que será la ventana.
frame = wx.Frame(None,title="Hola mundo")

#Mostramos la ventana.
frame.Show()

#Hacemos el el objeto app se ejecute en forma de bucle.
app.MainLoop();
```

Resultado:



Ejemplo 2

```
import wx

class MyFrame(wx.Frame):
    def __init__(self):
        super().__init__(parent=None, title='Hello World')
        panel = wx.Panel(self) #Creamos un panel
        my_sizer = wx.BoxSizer(wx.VERTICAL)
        # Creamos el objeto boxSizer para establecer los
        # widgets dentro de él
        self.text_ctrl = wx.TextCtrl(panel)
        my_sizer.Add(self.text_ctrl, 0, wx.ALL | wx.EXPAND, 5)
        my_btn = wx.Button(panel, label='Press Me')
        my_btn.Bind(wx.EVT_BUTTON, self.on_press)
        my_sizer.Add(my_btn, 0, wx.ALL | wx.CENTER, 5)
        panel.SetSizer(my_sizer)
        self.Show()

    def on_press(self, event):
        value = self.text_ctrl.GetValue()
        if not value:
            print("You didn't enter anything!")
        else:
            print(“You typed: "{value}"')

if __name__ == '__main__':
    app = wx.App()
    frame = MyFrame()
    app.MainLoop()
```

Ejemplo extraído de: https://realpython.com/python-gui-with-wxpython/

Explicación:

En este ejemplo se han utilizado medidores (Sizer), con el fin de crear diseños dinámicos. Estos últimos lo que hacen es administrar la ubicación de los widgets por nosotros, de modo que la interfaz no se vea afectada cuando el tamaño de la misma cambie.

En este ejemplo hemos declarado un *BoxSizer* al cual le hemos establecido que la orientación de agregado de items sea *wx.VERTICAL*. Por lo tanto, los widgets se establecerán de arriba hacia abajo.

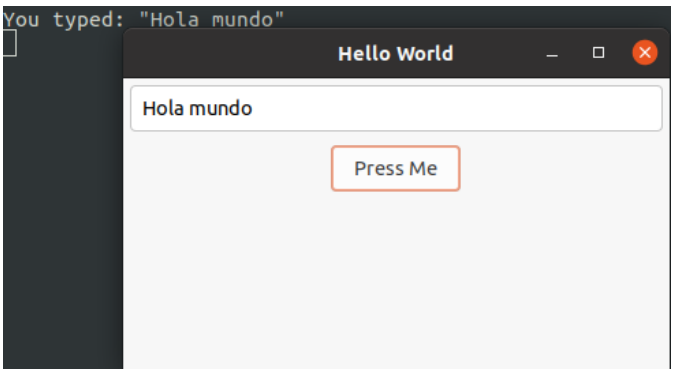
Para agregar un elemento a un sizer se realiza un .Add():

- Primer paŕametro widget a añadir.
- Segundo parámetro proporción de espacio respecto a otros widgets.
- Tercer parámetro, los flag que deseamos, por ejemplo en nuestro caso le decimos que establezca un recuadro al widget y que se expanda toda la pantalla.
- Cuarto parámetro, tamaño del borde del widget.

Para añadir el botón se realiza de la misma forma. añadiendolo al BoxSizer, tras realizar toda la configuración.

En cuanto a la configuración, cuando pulsemos sobre el botón queremos que se ejecute una acción la cual será mostrar por terminal un mensaje. Por lo tanto, tenemos que asignarle la acción Bind() con el evento del ratón, la cual será *wx.ON_BOTTON* asociándolo el evento la función *on_press()* con los parámetros self y event por convención.

Resultado:



Bibliografía

1. https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html
2. https://realpython.com/python-pyqt-gui-calculator/
3. https://unipython.com/desarrolla-tu-gui-con-wxpython-en-python-3/#:~:text=%C2%BFC%C3%B3mo%20surgi%C3%B3%20wxPython%3F,para%20comenzar%20con%20el%20proyecto.
4. https://es.wikipedia.org/wiki/WxPython
5. https://www.wxpython.org/pages/history/
6. https://www.wxwidgets.org/about/history/
7. https://realpython.com/python-gui-with-wxpython/
8. https://wxpython.org/pages/overview/
9. https://stackabuse.com/creating-python-gui-applications-with-wxpython/

10. https://www.askpython.com/python-modules/top-best-python-gui-libraries