Estudio de librerías para hacer interfaces gráficas

Alonso Bueno Herrero
Bartolomé Zambrana Pérez

UNIVERSIDAD
DE GRANADA

Contenidos

1. Introducción
2. Estudio de interfaces gráficas
    2.1. Java *Swing*
    2.2. Tk en Python: *Tkinter*
    2.3. QT en Python: *PyQT5*
    2.4. Wx en Python: *wxPython*

Contenidos

1. **Introducción**
2. Estudio de interfaces gráficas
    - 2.1. Java *Swing*
    - 2.2. Tk en Python: *Tkinter*
    - 2.3. QT en Python: *PyQT5*
    - 2.4. Wx en Python: *wxPython*

¿Qué librerías hemos escogido? ¿Por qué?

- Java **Swing**
- **TKinter** en Python
- **PyQT5**
- **wxPython**

Contenidos

1. Introducción
2. **Estudio de interfaces gráficas**
   - 2.1. Java *Swing*
   - 2.2. Tk en Python: *Tkinter*
   - 2.3. QT en Python: *PyQT5*
   - 2.4. Wx en Python: *wxPython*

Contenidos

1. Introducción
2. Estudio de interfaces gráficas
   - **2.1. Java *Swing***
   - 2.2. Tk en Python: *Tkinter*
   - 2.3. QT en Python: *PyQT5*
   - 2.4. Wx en Python: *wxPython*

Java Swing

- Orígenes: AWT

- AWT: diversos problemas
  - Depende de la plataforma
  - No sigue MVC
  - Componentes menores
  - No admite ***pluggable look and feel***

- **1997 → Swing**

¿Qué aporta Swing?

➔ independiente de Plataforma

➔ Sigue MVC

➔ Componentes más potentes

➔ Admite pluggable look and feel

Java Swing: Componentes & Contenedores

COMPONENTE

Es un control visual independiente.

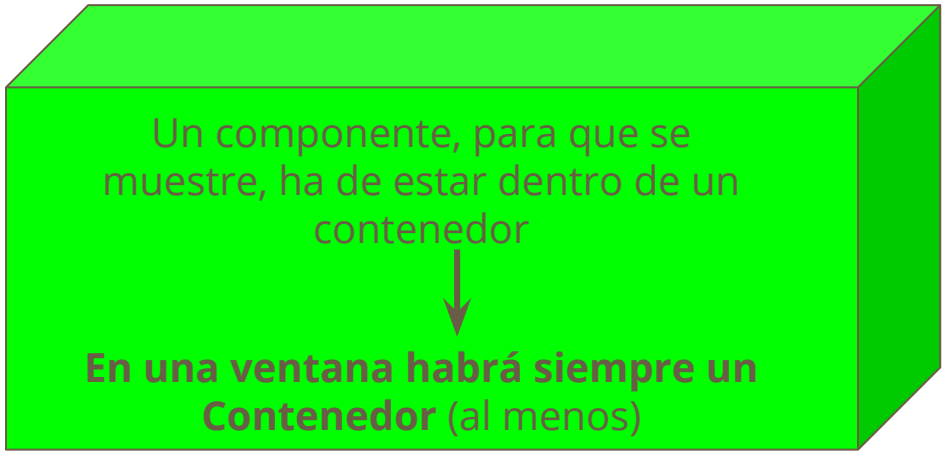
Ejemplos:

- un botón
- un campo de texto

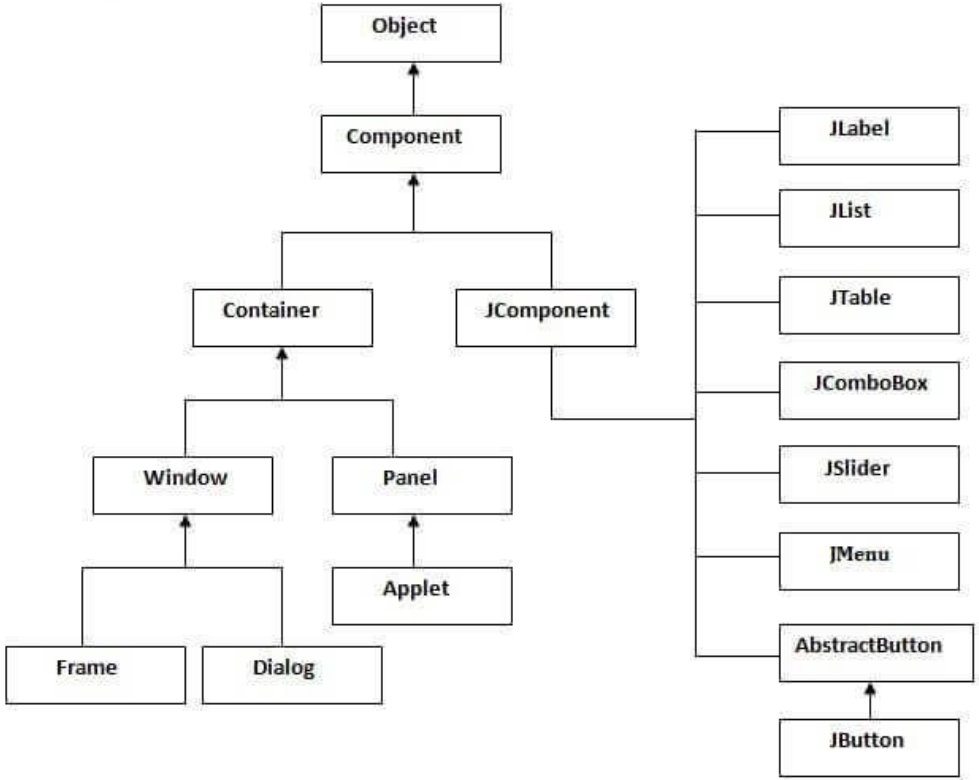
CONTENEDOR

Es un tipo especial de componente que **contiene** varios componentes.

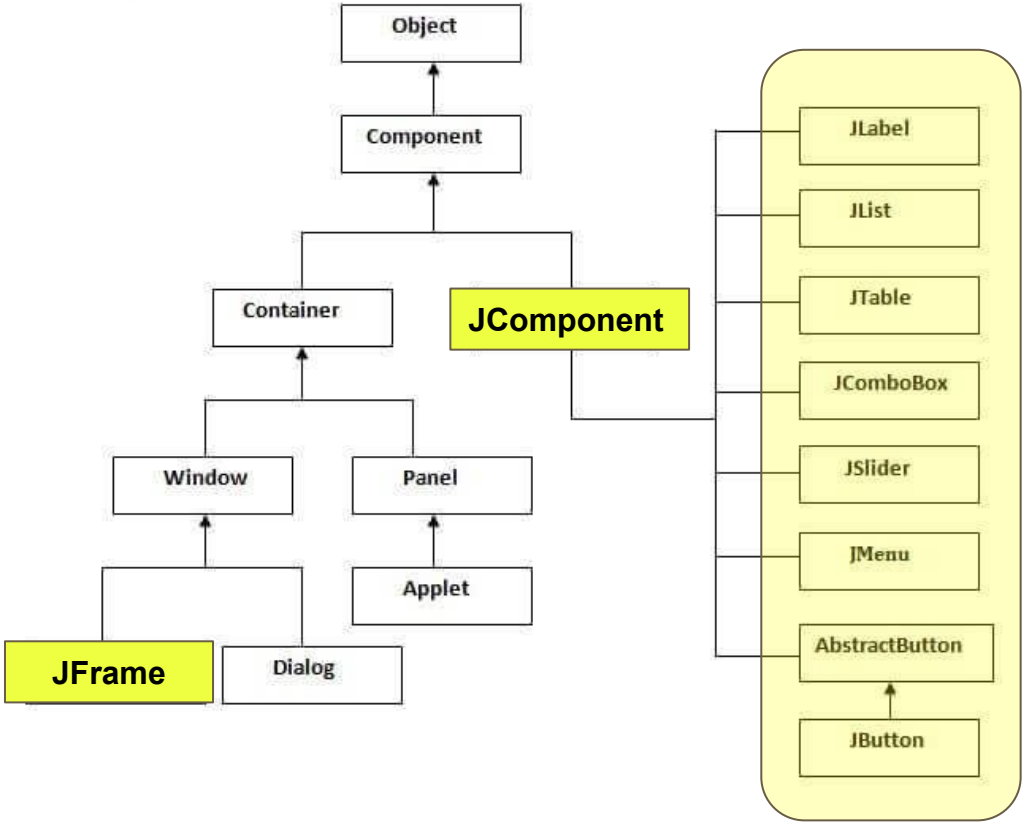
Dos tipos: nivel ***superior*** e ***inferior***.



Jerarquía de clases

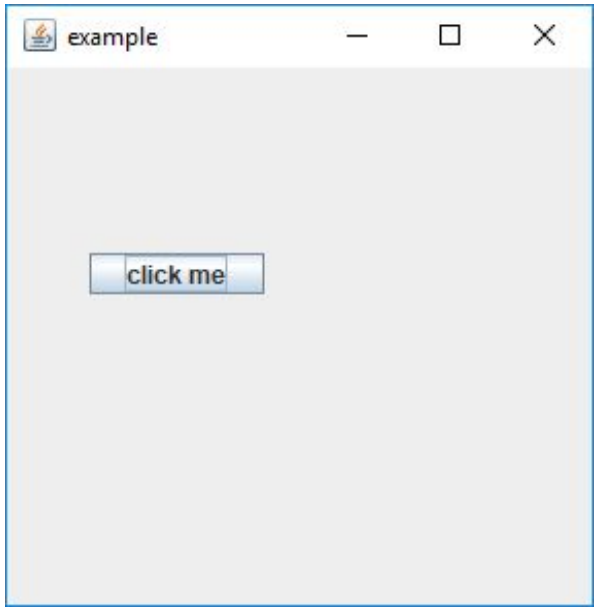


Jerarquía de clases

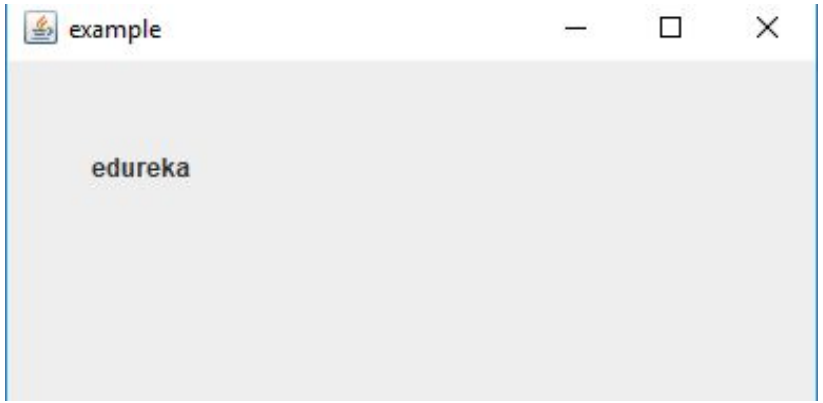


Principales componentes

Clase JButton

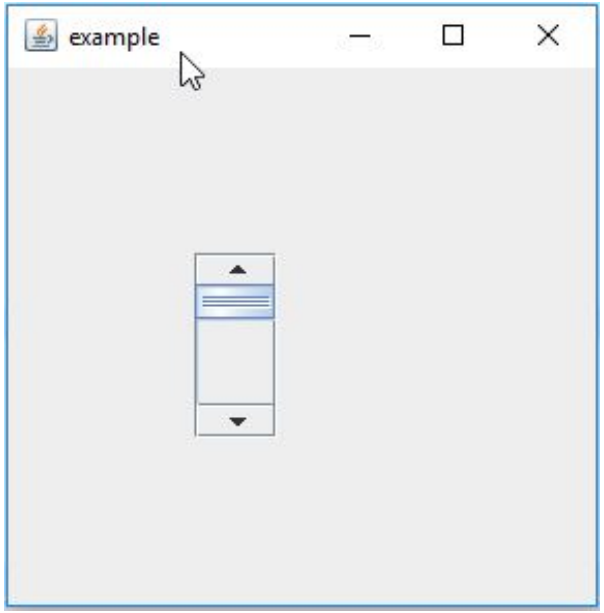


Clase JLabel

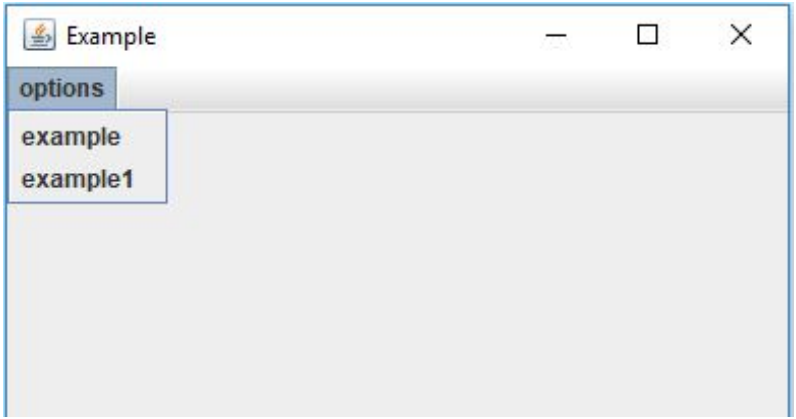


Principales componentes

Clase JScrollBar



Clase JMenu

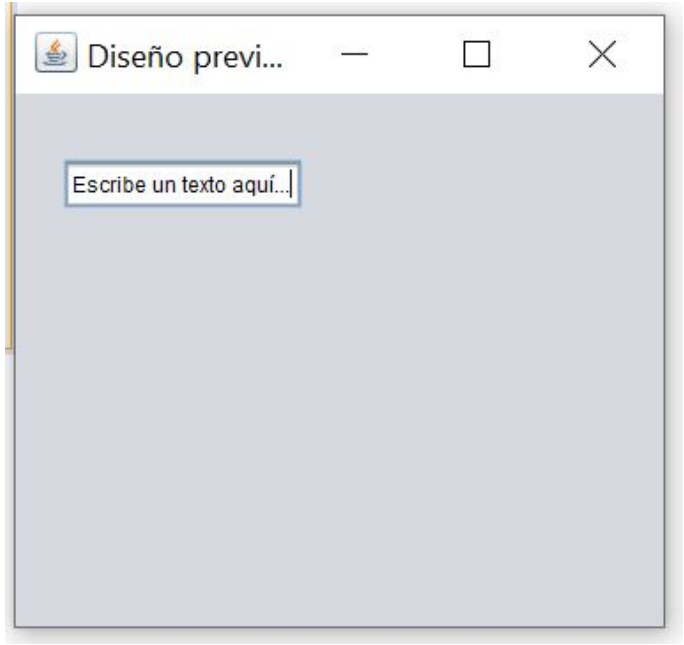


Principales componentes

¿Diferencia con una etiqueta (label)?

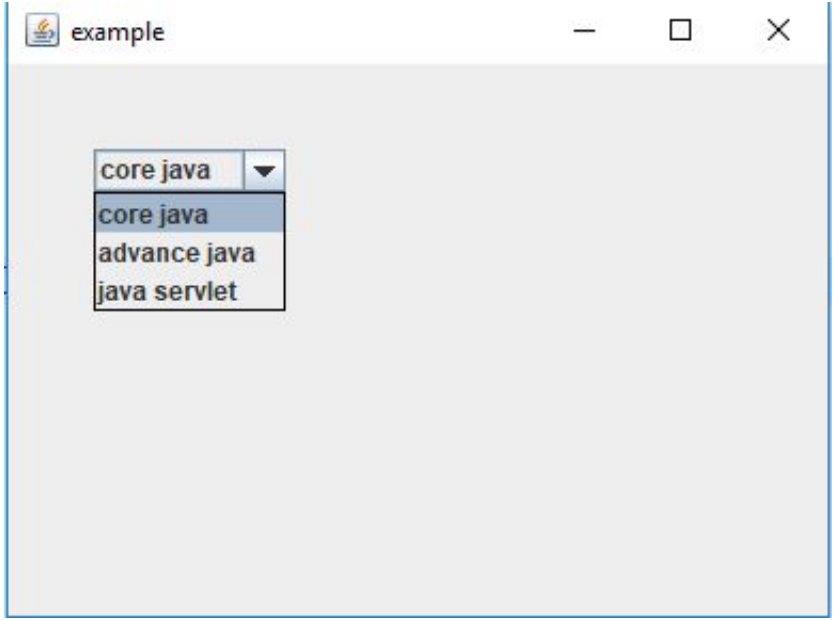
Clase JList

Clase JTextField



Principales componentes

Clase JComboBox



Ejemplo 1

```
import javax.swing.*;

public class Principal {
    private static void createAndShowGUI() {

        JFrame frame = new JFrame("Ventana Principal");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hola a todos.");
        frame.getContentPane().add(label);  // 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
            }
        });
    }
}
```

- Ejemplo ilustrativo

- Lo más recomendable es implementar la ventana como una herencia de **JFrame**.

Ejemplo 2

1. Ilustramos el uso básico del asistente de diseño de **Netbeans** para Swing.

2. ¿Cómo crear la primera ventana?

3. ¿Cómo acceder al código o a la vista de diseño?

4. ¿Cómo gestionar un evento [pulsación de un botón]?

Contenidos

1. Introducción
2. Estudio de interfaces gráficas
   - 2.1. Java *Swing*
   - **2.2. Tk en Python: *Tkinter***
   - 2.3. QT en Python: *PyQT5*
   - 2.4. Wx en Python: *wxPython*

¿Qué es TKinter?

- Es un ***binding*** de la biblioteca gráfica Tcl/Tk para **Python**
- Estándar para elaboración de GUIs en Python
    - Se instala al instalar Python (software libre)

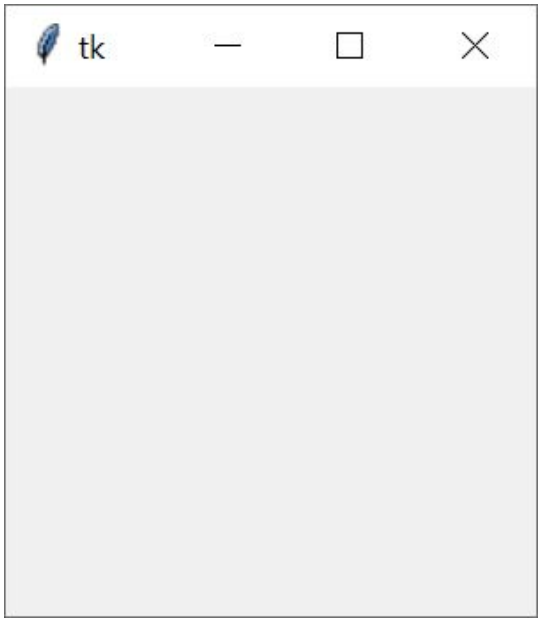
Componentes* en TKinter: *widgets

- Equivalente a los *componentes* de Swing en Python (se verá también en otras librerías posteriors)
- Construcción de ventana = ***widgets* jerarquizados**

Pasamos a ver ya los componentes principales...

Componentes principales

Tk → ventana vacía



Frame

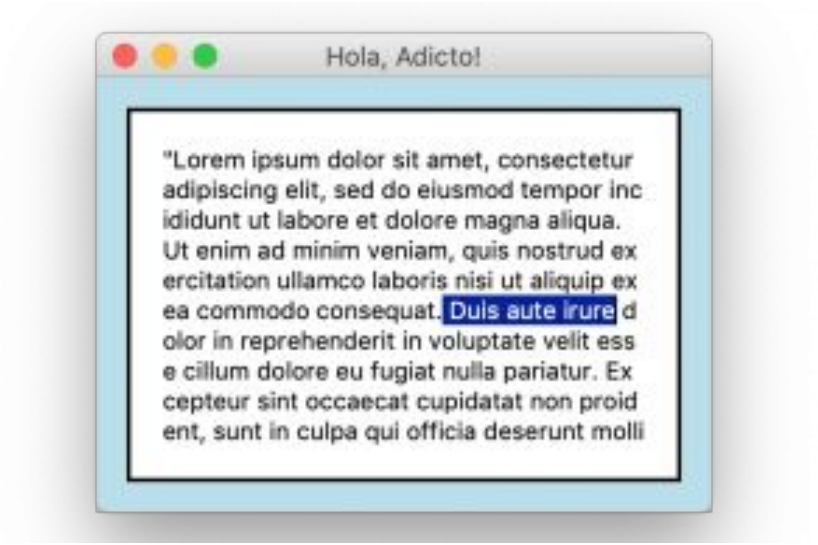
Label


Hola, Adicto!
¿Cómo estás?
Dale caña a Tkinter!

Entry


Hola, Adicto!
Te llamas...
Gwydion
Tu color favorito es...
azul
Tu contraseña de siempre es...
???????

Text

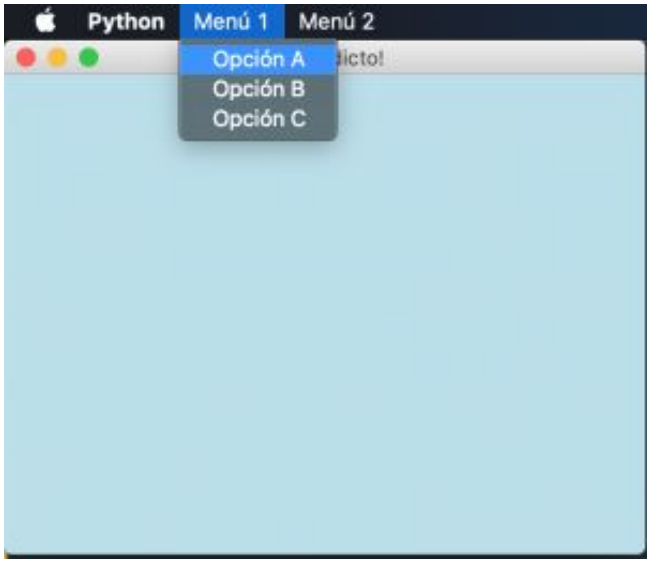

Hola, Adicto!
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor inc ididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud ex ercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure d olor in reprehenderit in voluptate velit ess e cillum dolore eu fugiat nulla pariatur. Ex cepteur sint occaecat cupidatat non proid ent, sunt in culpa qui officia deserunt molli

Button


Hola, Adicto!
Nombre
sdgsaga
Borrar

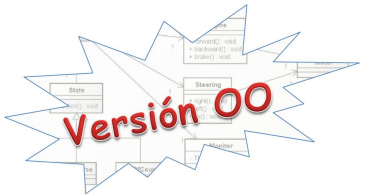
Checkbutton

Menu



Todos los widgets: https://guia-tkinter.readthedocs.io/es/develop/chapters/6-widgets/6.1-Intro.html

Ejemplo 1: Seleccionar archivo del Sistema de Archivos

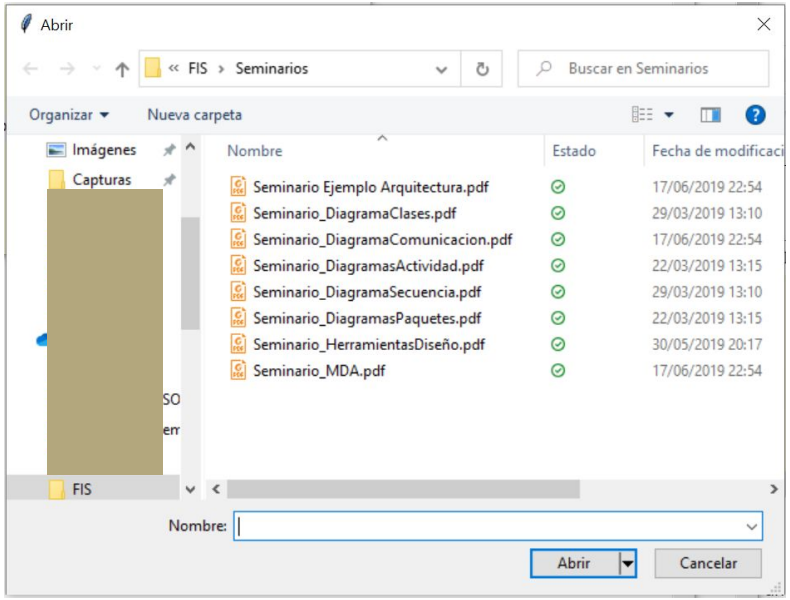



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk
from tkinter.filedialog import askopenfilename  # para 'Seleccionar archivo'

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
```

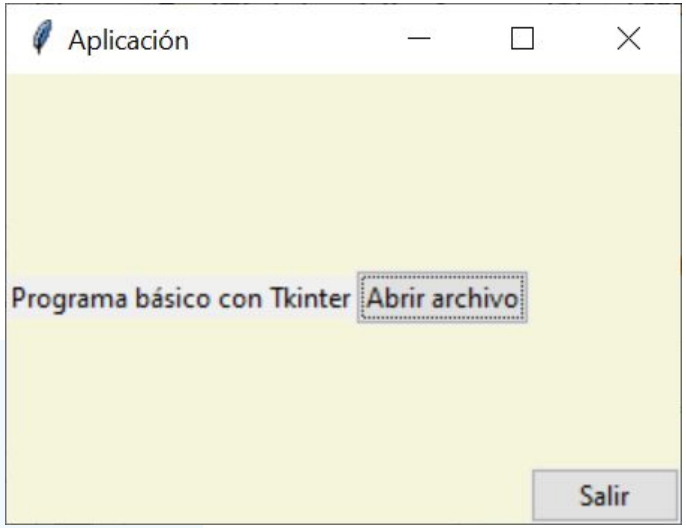
Ejemplo 1

Función para implementar *"Seleccionar archivo"*:

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

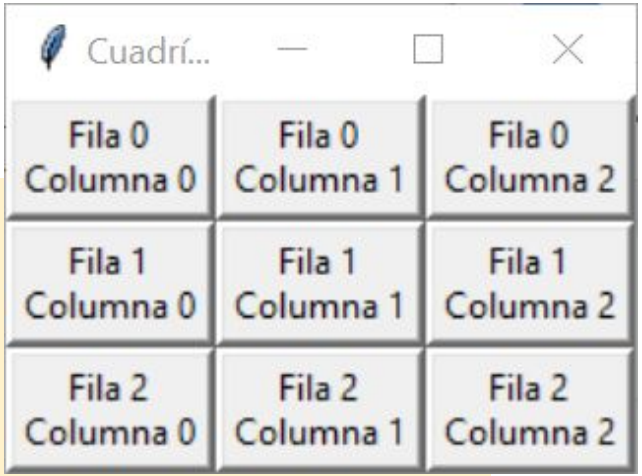


Ejemplo 1: Resto de código. Ventana principal




```
def main():
    mi_app = Aplicacion()
    return 0


# __name__ es una variable especial de python que
constituye el main del programa
if __name__ == '__main__':
    main()
```

Ejemplo 2: Cuadrícula de botones




```
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
```

Contenidos

1. Introducción
2. Estudio de interfaces gráficas
   - 2.1. Java *Swing*
   - 2.2. Tk en Python: *Tkinter*
   - **2.3. QT en Python: *PyQT5***
   - 2.4. Wx en Python: *wxPython*

PyQt5: Introducción.

ORÍGENES

1999 → aparece como un binding de Qt, creado por *Riverbank Computin.*

Es un conjunto de bibliotecas C++ *multiplataforma* que conforman una API de alto nivel.

Se puede decir que es un conjunto de enlaces completos a *QT5 for Python.*

Licencia

Se publicó con licencia GPL v3. → Permite el desarrollo de aplicaciones propietarias.

Licencia compatible con Qt, aunque Qt tiene licencia LGPL

PyQt5: Componentes.

Cuando instalamos PyQt5 se instalan todos los módulos. Ejemplos:

- QtWidgets.
- QtCore.
- Qt3DAnimation.

Podemos visualizar todos los módulos en:
https://www.riverbankcomputing.com/static/Docs/PyQt5/module_index.html#ref-module-index

Cuenta además con un ***complemento*** que posibilita la creación de interfaces desde **Qt Designer**.

PyQt5: Conceptos básicos.

Desarrollo eficiente de interfaces gráficas: necesitamos saber qué es...

1. **Aplicación.**
2. **Bucle de Eventos.**
3. **Ventanas**
4. **Widgets**
5. **Medidores.**
6. **Diálogos**
7. **Eventos y Bucle de Eventos**
8. **Señales y Ranuras**

PyQt5: Conceptos básicos

Aplicación

Siempre hay que crear un objeto *QAplication*

Ventana

Permite la interacción con el usuario. Para poder utilizarlo se ha detener el widget central creado.

Bucle de eventos

Bucle infinito que *espera* a que ocurran los eventos en la GUI.


```
#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

#Empezamos la aplicación
app = QApplication(sys.argv)

#Ventana Central
window = QWidget()
window.setWindowTitle('Hola Mundo Aplicación')
window.move(60,15)

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_())
```

PyQt5: Conceptos básicos

Widgets

Son los componentes de la interfaz.

Los más básicos y usuales son los siguientes:

- ***QPushBotton***.
- ***QLabel.***
- ***QLineEdit.***
- ***QComboBox.***

Medidores.

Conserva el diseño original al redimensionar la ventana.

- ***QHBoxLayout.***
- ***QVBoxLayout.***
- ***QGridLayout.***
- ***QFormLayout.***

PyQt5: Conceptos básicos

Diálogos

Para implementarlos hay que crear una clase que herede de *QDialog*.

Ejemplo:

https://files.realpython.com/media/dialog-style-app.d453259dee67.png

Eventos, señales y ranuras

Ejemplos

- Detección del movimiento de ratón.
- Click en un botón

Siguen el esquema de 1 ⟶ M (Una señal muchas ranuras).

Puede haber señales conectadas en serie.

¿Cómo asociar un evento?

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

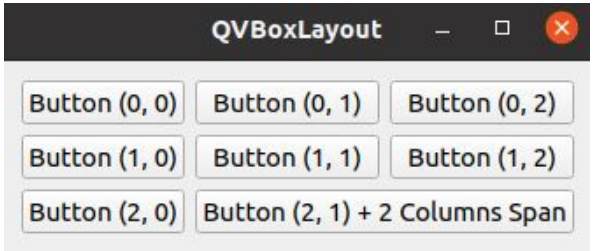
Ejemplo:

```
btn.clicked.connect(funcionMensaje)
```

PyQt5: Ejemplo 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


app = QApplication(sys.argv)

window = QWidget()
window.setWindowTitle('QVBoxLayout')
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)
```

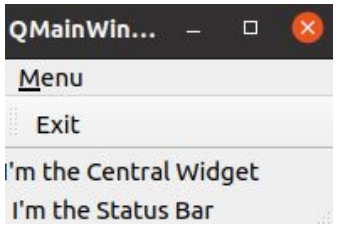
```
#Establecemos el layout para la adición de
widgets, en este caso botones.

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)


window.show()
sys.exit(app.exec_())
```



PyQt5: Ejemplo Menú

```
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_())
```



Contenidos

1. Introducción
2. Estudio de interfaces gráficas
   - 2.1. Java *Swing*
   - 2.2. Tk en Python: *Tkinter*
   - 2.3. QT en Python: *PyQT5*
   - **2.4. Wx en Python: *wxPython***

wxPython: Introducción.

ORÍGENES

Finales años 90 → fusión de wxWidgets y Python para desarrollar GUI compatible con Windows3.1 y HP-UX.

2012 → Proyecto Phoenix compatible Python3: corrige problemas de mantenimiento y velocidad originales

Casos de éxito:

→ **Trastana**. Herramientas de análisis multimedia.

→ Vintech RCAM-Pro.

→ Robot Framework IDE. Editor de datos de prueba de Robot Framework.

→ Entre muchos otros casos...

wxPython: Conceptos básicos.

Desarrollo eficiente de interfaces gráficas: necesitamos saber qué es...

1. **Aplicación**
2. **Ventana**
3. **Bucle de Eventos.**
4. **Widgets.**
5. **Posicionamiento Absoluto y Medidores.**
6. **Eventos.**

wxPython: Conceptos básicos

Aplicación

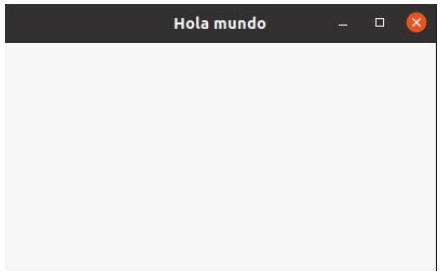
Siempre hay que crear un objeto wx.App()

Ventana

Permite la interacción con el usuario

Bucle de eventos

Bucle infinito que *espera* a que ocurran los eventos en la GUI.




```
#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();
```

wxPython: Conceptos básicos

Widgets

Son los componentes de la interfaz.

Los más básicos y usuales son los siguientes:

- ***wx.Panel***.
- **wx.*TextCtr***.
- ***wx.Button***.

Posicionamiento Absoluto

Pueden surgir problemas con el reajuste de la ventana.

Posicionamiento Ajustado (*Medidores*)

Conserva el diseño original al redimensionar la ventana.

- ***wx.BoxSizer.***
- ***wx.GridSizer.***
- ***wx.FlexGridSizer.***

wxPython: Conceptos básicos

Eventos

Acciones tras la detección de alguna acción del usuario. *Ejemplo*: pulsación de un botón

¿Cómo asociar el evento?

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

Convenio

Para la función que se ha de pasar como argumento hay que definir los siguientes parámetros:

- *self:* instancia del objeto que definimos.
- event: evento

```
function(self,event):
    ...
    ...
```

wxPython: Ejemplo final.

```
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(f"You typed: "{value}"")

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

wxPython: Ampliación.

Para ampliar y profundizar más en el diseño con esta librería recomendamos los tutoriales disponibles en

https://zetcode.com/wxpython/