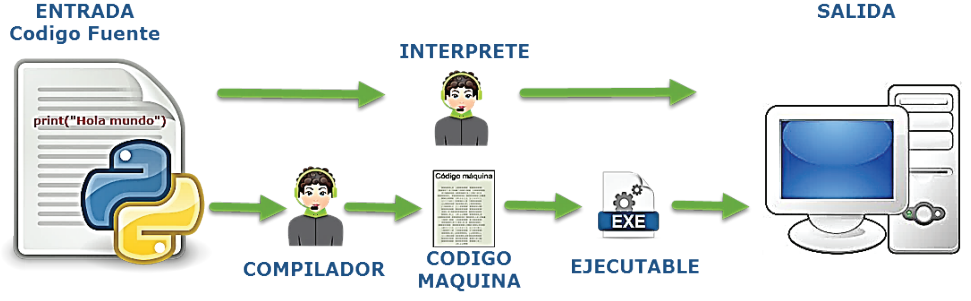
INTRODUCCIÓN

Python es un lenguaje de programación de propósito general creado por Guido Van Rosum en los 90 trabajo en Google y en la actualidad en Dropbox, su nombre proviene del cómic Monty Python. Cuenta con una sintaxis muy limpia y legible. Posee tipado dinámico esto quiere decir que una variable puede poseer datos de varios tipos, junto con su naturaleza interpretada, hacen de este un lenguaje para ser el primer en aprender. Python es un lenguaje interpretado, lo que nos indica que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas.



Python está escrito en el lenguaje C, por lo que se puede extender a través de su api en C o C++ y escribir nuevos tipos de datos, funciones, etc. En la actualidad hay dos vertientes la versión 2.x y 3.x, al final llegara el momento que se integraran estas dos versiones, es recomendable utilizar la última versión estable 3.x Algunas de las características más importantes es que Python es multiparadigma: Programación estructurada, Programación Orientada a Objetos y Programación Funcional.

Python en el desarrollo web se puede utilizar los framework: Django y Flask. Entre las empresas más conocidas que utilizan Python tenemos: Dropbox y Instagram.

En Data Sciencia y Machine Learning tenemos: Pandas, Scikit-Learn y TensorFlow. Además, es multiplataforma: Linux, Windows, Mac OS, Solaris, etc.

Python permite ser implementado en diferentes lenguajes:

- CPython - Python tradicional escrito en C.
- Jython - Python para la JVM.
- IronPython - Python para .NET.
- Pypy - Python más rápido con compilador JIT.
- StacklessPython - Branch de CPython con soporte para microthreads.

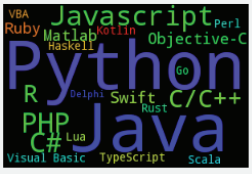
A continuación, un listado de los lenguajes más populares:

PYPL PopularitY of Programming Language

The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.

The more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from Google Trends.

If you believe in collective wisdom, the PYPL Popularity of Programming Language index can help you decide which language to study, or which one to use in a new software project.



Worldwide, Jun 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1	↑	Python	23.04 %	+5.2 %
2	↓	Java	22.45 %	-0.6 %
3	↑↑	Javascript	8.6 %	+0.3 %
4	↓	PHP	8.21 %	-1.6 %
5	↓	C#	8.01 %	-0.4 %
6		C/C++	6.15 %	-1.1 %
7	↑	R	4.14 %	+0.1 %
8	↓	Objective-C	3.46 %	-1.0 %
9		Swift	2.75 %	-0.8 %
10		Matlab	2.15 %	-0.4 %
11		Ruby	1.7 %	-0.4 %
12	↑↑↑	TypeScript	1.54 %	+0.4 %
13		VBA	1.37 %	-0.0 %
14		Scala	1.23 %	-0.1 %
15	↓↓↓	Visual Basic	1.22 %	-0.2 %
16	↑↑↑↑↑↑	Kotlin	0.93 %	+0.6 %
17		Go	0.92 %	+0.3 %

Fuente: http://pypl.github.io/PYPL.html

Este libro te introduce de manera fácil los conceptos y características básicas, intermedias y avanzadas del lenguaje de Python. Es bueno tener un intérprete de Python a mano para experimentar.

1

KIVY

1.1 QUÉ ES KIVY

Kivy es una librería open source para el desarrollo rápido de aplicaciones equipadas con novedosas interfaces de usuario, como aplicaciones multitáctiles.

Filosofía

A continuación, describiremos que es Kivy y de qué se trata:

- **Fresh.-** Kivy está hecho para el presente y futuro. Los nuevos métodos de entrada como Multi-Touch se han vuelto cada vez más importantes para la interacción.
- **Fast.-** Kivy es rápido. Esto se aplica tanto al desarrollo de la aplicación como a la velocidad de ejecución de la misma. Hemos optimizado Kivy ha sido optimizado de muchas maneras:

- Implementamos funcionalidad de tiempo crítico en el nivel C para aprovechar el poder de los compiladores existentes. Más importante aún, también utilizamos algoritmos inteligentes para minimizar los costos.
- También usamos la GPU donde tiene sentido en nuestro contexto. El poder computacional de las tarjetas gráficas de hoy en día supera a la de las CPU de hoy en día para algunas tareas y algoritmos, especialmente el dibujo. Es por eso que tratamos de dejar que la GPU haga la mayor parte del trabajo posible, aumentando así rendimiento considerablemente.

◤ **Flexible.-** Kivy es flexible. Esto significa que se puede ejecutar en una variedad de dispositivos diferentes, incluido Android.

Teléfonos inteligentes y tabletas. Admitimos todos los principales sistemas operativos (Windows, Linux, OS X). Siendo flexible.

También significa que el rápido desarrollo de Kivy le permite adaptarse rápidamente a las nuevas tecnologías. Más que una vez hemos agregado soporte para nuevos dispositivos externos y protocolos de software, a veces incluso antes fueron liberados.

◤ **Focused.-** Kivy está enfocado. Puede escribir una aplicación simple con algunas líneas de código. Se crean los programas Kivy utilizando el lenguaje de programación Python, que es increíblemente versátil y potente, pero fácil de usar.

◤ **Funded.-** Kivy es una solución con influencia de la comunidad, desarrollada profesionalmente y con respaldo comercial.

◤ **Free.-** Kivy es free o uso gratuito.

1.2 INSTALACIÓN DE KIVY EN WINDOWS

Para utilizar Kivy, Necesitamos tener instalado previamente Python.

En línea de comando escriba lo siguiente:

1. Asegúrese la última versión de: pip y wheel:

```
python -m pip install --upgrade pip wheel setuptools
```

2. Instale las dependencias:

```
python -m pip install docutils pygments
pypiwin32 kivy.deps.sdl2 kivy.deps.glew
python -m pip install kivy.deps.gstreamer
```

3. Para Python 3.5, ofrecemos angle que se puede usar en lugar de glew y se puede instalar con:

```
python -m pip install kivy.deps.angle
```

4. Instalación de kivy:

```
python -m pip install kivy
```

Opcionalmente, Instalación de Ejemplos:

```
python -m pip install kivy_examples
```

1.3 INSTALACIÓN DE KIVY EN OS X

Usando Homebrew.

1. Requisitos usando homebrew:

```
$ brew install pkg-config sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer
```

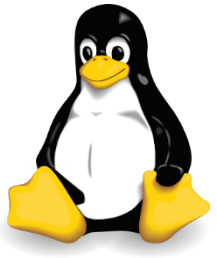
2. Instalando con Cython y Kivy usando pip:

```
$ pip install Cython==0.26.1
$ pip install kivy
```

Para instalar la versión de desarrollo:

```
$ pip install https://github.com/kivy/kivy/archive/master.zip
```

1.4 INSTALACIÓN EN LINUX



Para instalar paquetes relativos de distribución .deb/.rpm/…

1. Agregar uno de los PPA.

stable builds:	$ sudo add-apt-repository ppa:kivy-team/kivy
nightly builds:	$ sudo add-apt-repository ppa:kivy-team/kivy-daily

2. Actualice su lista de paquetes.

```
$ sudo apt-get update
```

3. Instalar Kivy.

Python2 - python-kivy:

```
$ sudo apt-get install python-kivy
```

Python3 - python3-kivy:

```
$ sudo apt-get install python3-kivy
```

Opcionalmente los ejemplos:

```
$ sudo apt-get install python-kivy-examples
```

Debian (Jessie or Newer)

1. Agregue uno de los PPA a tus fuentes apt manual o con mediante Synaptic.

stable builds:	deb http://ppa.launchpad.net/kivy-team/kivy/ubuntu xenial main
daily builds:	deb http://ppa.launchpad.net/kivy-team/kivy-daily/ubuntu xenial main

2. Agregue la clave GPG a su apt keyring ejecutando.

```
as user:
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys A863D2D6
as root:
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys A863D2D6
```

Linux Mint

1. Ubica tu versión de Ubuntu en el cual está basada tu instalación.
2. Continua con tu instalación.

Bodhi Linux

1. Ubica tu versión de distribución que está ejecutando y use la tabla a continuación:

Bodhi 1:	Ubuntu 10.04 LTS aka Lucid (No packages, just manual install)
Bodhi 2:	Ubuntu 12.04 LTS aka Precise
Bodhi 3:	Ubuntu 14.04 LTS aka Trusty
Bodhi 4:	Ubuntu 16.04 LTS aka Xenial

OpenSuSE

1. Para instalar kivy vaya la a url http://software.opensuse.org/package/python-Kivy.
2. Si deseas acceder a los ejemplos selecciones **python-Kivy-examples.**

Fedora

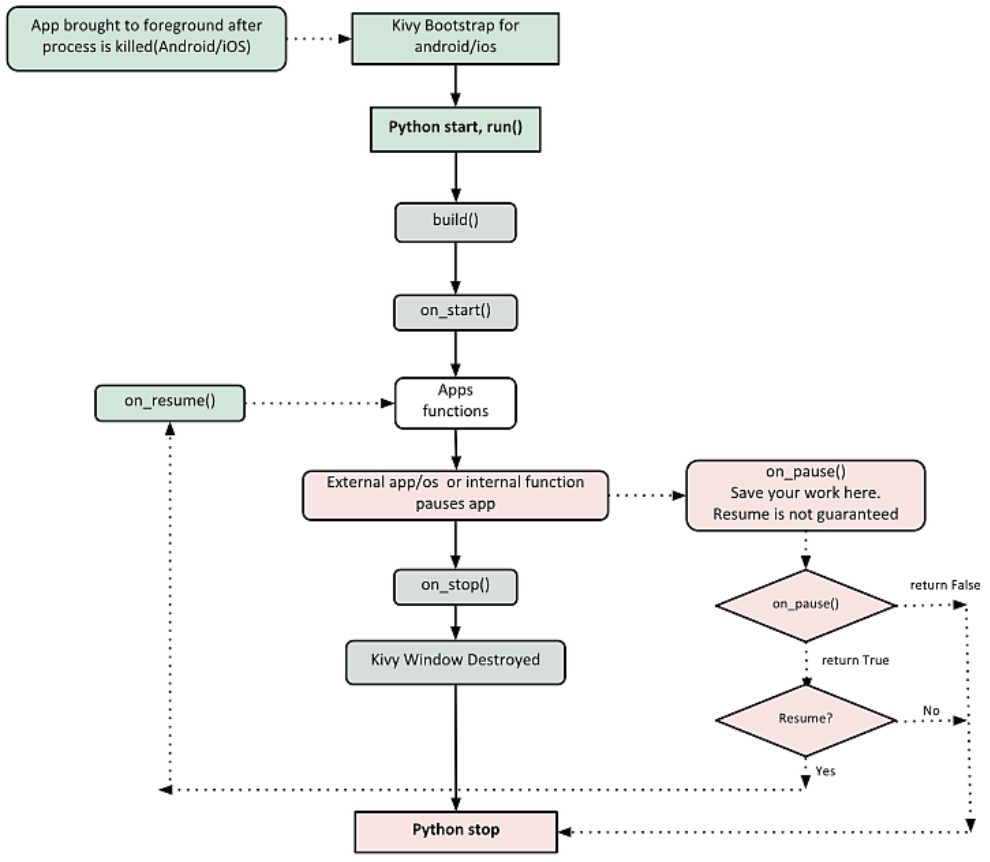
1. Agregue al repositorio a través del terminal:

```
Fedora 18
$ sudo yum-config-manager  --add-repo=http://download.opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_18/home:thopiekar:kivy.repo
```

```
Fedora 17
$ sudo yum-config-manager --add-repo=http://download.opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_17/home:thopiekar:kivy.repo
```

```
Fedora 16
$ sudo yum-config-manager --add-repo=http://download.opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_16/home:thopiekar:kivy.repo
```

1.5 CICLO DE VIDA DE UNA APLICACIÓN KIVY



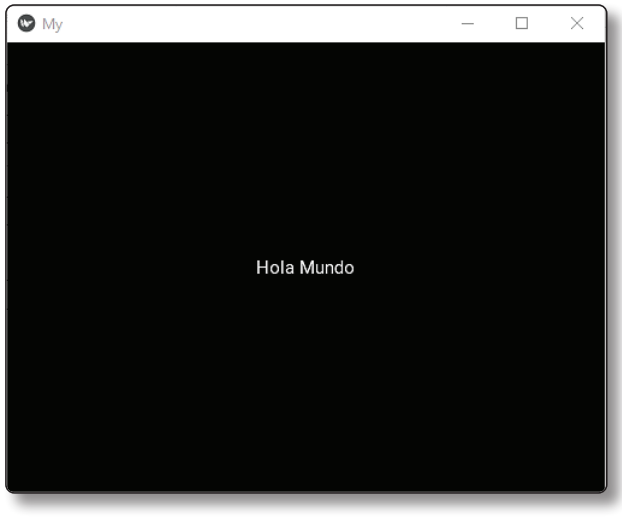
1.6 MI PRIMERA APLICACIÓN

Ahora vamos a crear el famoso HOLA MUNDO:

HolaMundo.py

```
import kivy
kivy.require('1.10.0')
from kivy.app import App
from kivy.uix.button import Label
class MiAplicacion (App):
    def build(self):
        return Label(text='Hola Mundo')
if __name__ == "__main__":
    MiAplicacion ().run()
```

Al ejecutarse:



A continuación, explicaremos el código línea a línea:

Indicamos la versión de Kivy.

```
import kivy
kivy.require('1.10.0')
```

Importamos la clase base de tu App desde app:

```
from kivy.app import App
```

El uix.button es la sección que contiene los elementos de la interfaz de usuario, como diseños y widgets que se desea mostrar.

```
from kivy.uix.button import Label
```

Aquí es donde estamos definiendo la clase base de nuestra aplicación Kivy. Solo deberías necesitar cambiar el nombre de su aplicación MyApp en esta línea:

```
class MiAplicacion(App):
```

Función que inicializa y retorna el widget.

```
def build(self):
    return Label(text='Hola Mundo')
```

La clase MyApp se inicializa y se llama a su método run (). Esto inicializa y comienza nuestro Kivy:

```
if __name__ == "__main__":
    MiAplicacion().run()
```

1.7 WIDGETS BÁSICOS

Un widget es el componente básico de las interfaces en Kivy. Una interfaz de usuario generalmente tiene muchos elementos, como cuadros de texto de entrada, etiquetas, listas desplegables, botones, botones de radio, etc. Estos elementos se denominan widgets en Kivy.

Un widget está representado por una subclase de la kivy.uix.widget.Widget clase.

Un widget puede tener propiedades como id, color, texto, tamaño de fuente, etc.

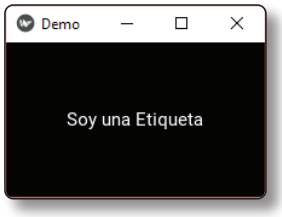
Un widget puede desencadenar algunos eventos como:

- touch down
- touch move
- touch up

A continuación, mostraremos el código kivy describiendo algunos widgets básicos:

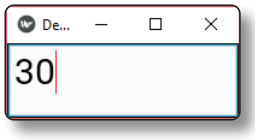
El nombre de la clase Label y sus propiedades en minúscula.

```
Label:
    id: etiqueta1
    text: 'Soy una Etiqueta'
    font_size: 15
```



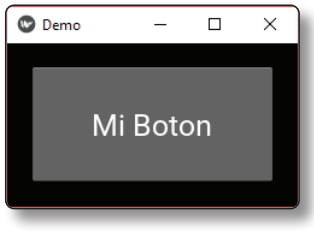
El nombre de la clase TextInput y sus propiedades en minúscula.

```
TextInput:
    id: numero1
    multiline:False
    font_size: 30
```



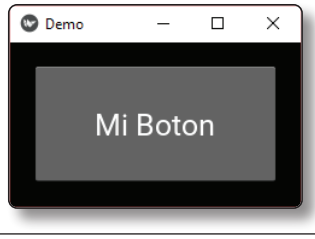
El nombre de la clase Button y sus propiedades en minúscula.

```
Button:
    id: boton1
    text: 'Mi Boton'
    font_size: 25
```



La clase Button tiene dos eventos adicionales: on_press similar a on_touch_down y on_release on_touch_down.

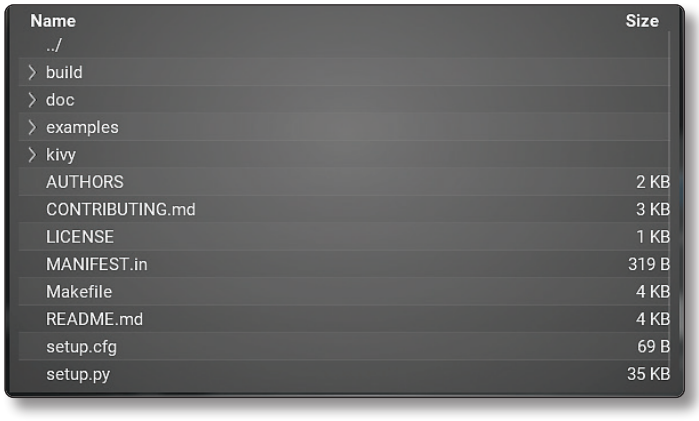
```
Button:
    id: boton1
    text: 'Mi Boton'
    font_size: 25
    on_press: root.accion1()
```

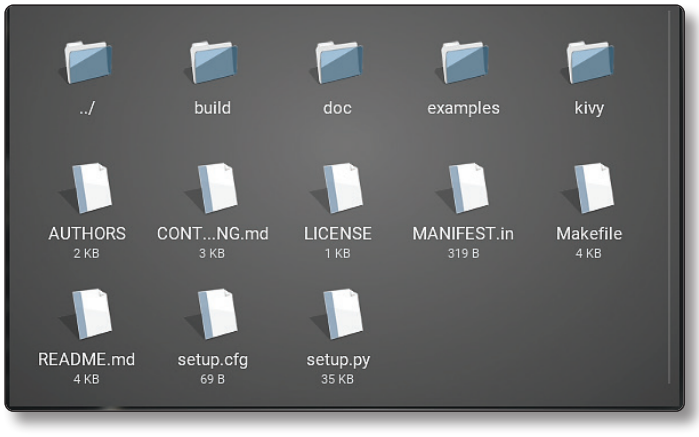


1.8 MÓDULO - FILECHOOSER

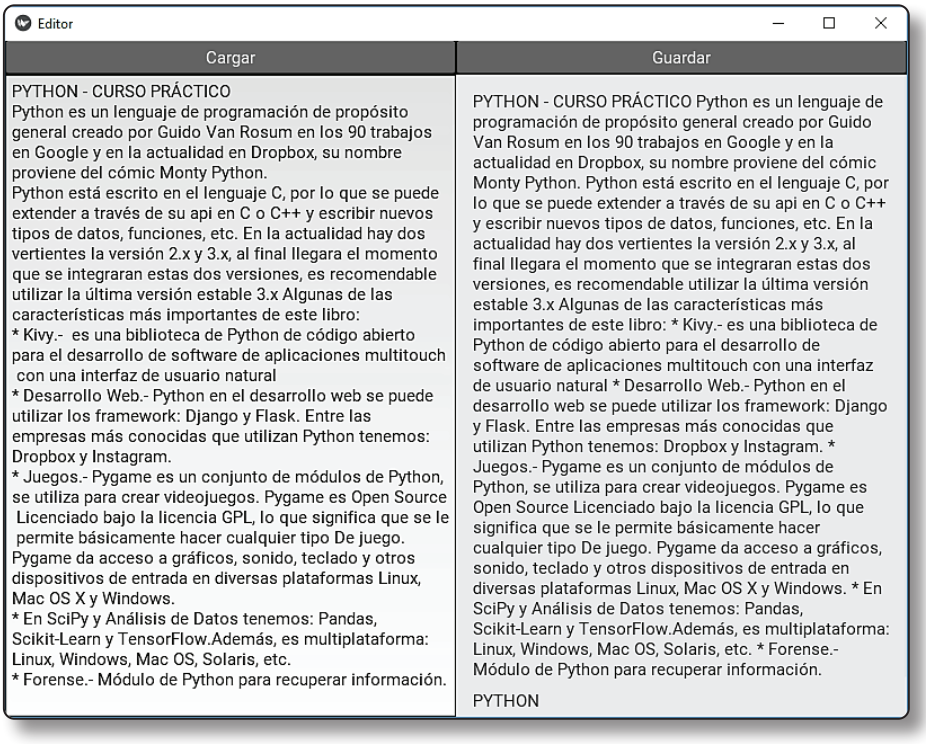
El módulo FileChooser proporciona varias clases para describir, visualizar y examinar archivos.

FileChooserListView:
id: filechooser





1.9 CONTENEDORES

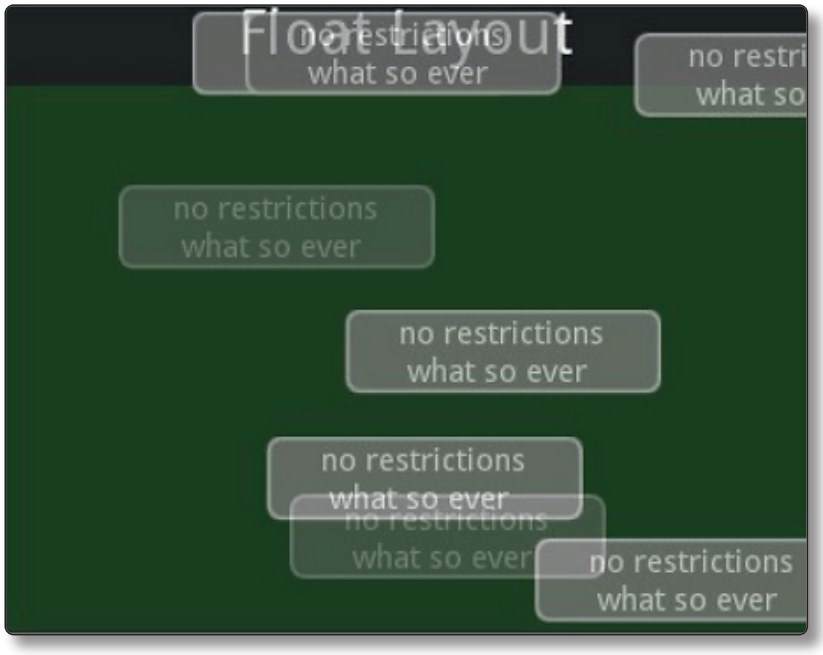


Un contenedor es donde colocamos los elementos de una aplicación o sitio web, los contenedores no ayudaran en el ordenamiento adecuado de los elementos que una aplicación puede contener gráficamente.

Kivy cuenta con muchos contenedores con características diferentes. Entre los que tenemos:

- FloatLayout
- Gridlayout
- Boxlayout
- Stacklayout
- Relavitelayout
- Anchorlayout

1.9.1 Floatlayout



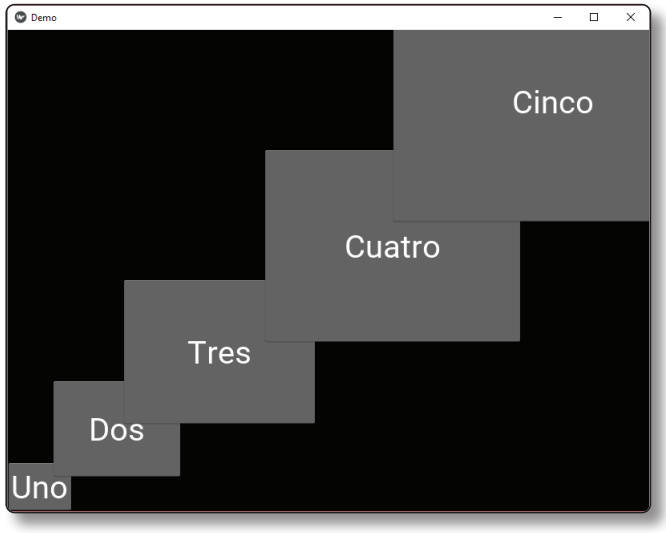
Este es uno de los contenedores más utilizados, gracias a la gran ventaja que tenemos de poner elementos en el lugar que más nos guste. Ejemplo:

FloatLayout1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.floatlayout import FloatLayout
Builder.load_string(‘’’
<Flotante>:
    Button:
        font_size: 40
        text: ‘Uno’
        size_hint: 0.1, 0.1
        pos_hint: {“x”: 0, ‘y’:0}
    Button:
        font_size: 40
        text: ‘Dos’
        size_hint: 0.2, 0.2
        pos_hint: {“x”: 0.07, ‘y’:0.07}
```

```
    Button:
        font_size: 40
        text: 'Tres'
        size_hint: 0.3, 0.3
        pos_hint: {"x": 0.18, 'y':0.18}
    Button:
        font_size: 40
        text: 'Cuatro'
        size_hint: 0.4, 0.4
        pos_hint: {"x": 0.4, 'y':0.35}
    Button:
        font_size: 40
        text: 'Cinco'
        size_hint: 0.5, 0.5
        pos_hint: {"x": 0.6, 'y':0.6}
''')
class Flotante(FloatLayout):
    pass
class DemoApp(App):
    def build(self):
        return Flotante()
if __name__ == '__main__':
    DemoApp().run()
```

Al ejecutarse:



1.9.2 Gridlayout

Para utilizar este layout tenemos que definir por lo menos su dimensión, si queremos que tenga una columna o más de igual forma que las filas, puedes ver este layout como una hoja de cálculo donde cada elemento toma su lugar de acuerdo al orden que los agregues, por ejemplo:

Gridlayout 1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.gridlayout import GridLayout
Builder.load_string('''
<Grid>:
    cols:3
    rows:3
    Button:
        font_size: 40
        text: 'Uno'
    Button:
        font_size: 40
        text: 'Dos'
    Button:
        font_size: 40
        text: 'Tres'
    Button:
        font_size: 40
        text: 'Cuatro'
    Button:
        font_size: 40
        text: 'Cinco'
    Button:
        font_size: 40
        text: 'Seis'
''')
class Grid(GridLayout):
    pass
class DemoApp(App):
    def build(self):
        return Grid()
if __name__ == '__main__':
    DemoApp().run()
```

Al ejecutarse:

1.9.3 Boxlayout

En este layout puedes agregar varios elementos los cuales se ordenan dependiendo del tamaño de cada elemento dentro del mismo.

Por ejemplo:

Boxlayout1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
Builder.load_string(‘’’
<caja>:
    orientation: ‘vertical’
    padding: 20
    BoxLayout:
        Button:
            text: “Primera Fila, Primera Columna”
            bold: True
        Button:
            text: “Primera Fila, Segunda Columna”
```

```
            color: [1, 0, 0, 1]
            bold: True
    BoxLayout:
        Button:
            text: “Segunda Fila, Primera Columna”
            color: [0, 0, 1, 1]
            bold: True
        Button:
            text: “Segunda Fila, Segunda Columna”
            color: [1, 1, 0, 1]
            bold: True
‘’’)
class Caja(BoxLayout):
    pass
class DemoApp(App):
    def build(self):
        return Caja()
if __name__ == ‘__main__’:
    DemoApp().run()
```

Al ejecutarse:

1.9.4 Stacklayout

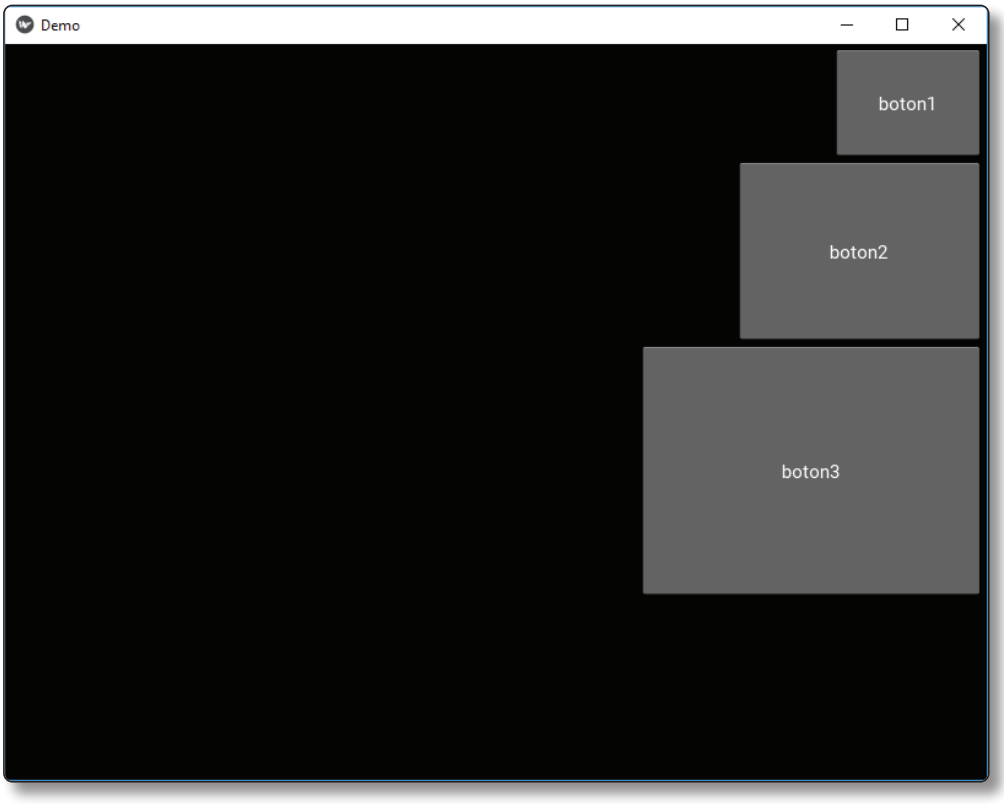
En este layout podemos agregar varios elementos cada elemento se acomoda dependiendo de su tamaño del espacio que el Stacklayout tenga para utilizar. Nos seria de mucha utilidad si queremos agregar varios elementos y deseamos que tengan el mismo tamaño. El layout le define el espacio a cada elemento dependiendo del espacio total que el layout tenga disponible.

Ejemplo:

Stacklayout 1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
Builder.load_string('''
<caja>:
    StackLayout:
        spacing: 5
        padding: 5
        orientation: 'tb-rl'
        Button:
            text: "boton1"
            size_hint: 0.15, 0.15
        Button:
            text: "boton2"
            size_hint: 0.25, 0.25
        Button:
            text: "boton3"
            size_hint: 0.35, 0.35
''')
class Caja(BoxLayout):
    pass
class DemoApp(App):
    def build(self):
        return Caja()
if __name__ == '__main__':
    DemoApp().run()
```

Al ejecutarse:



1.9.5 Relativelayout

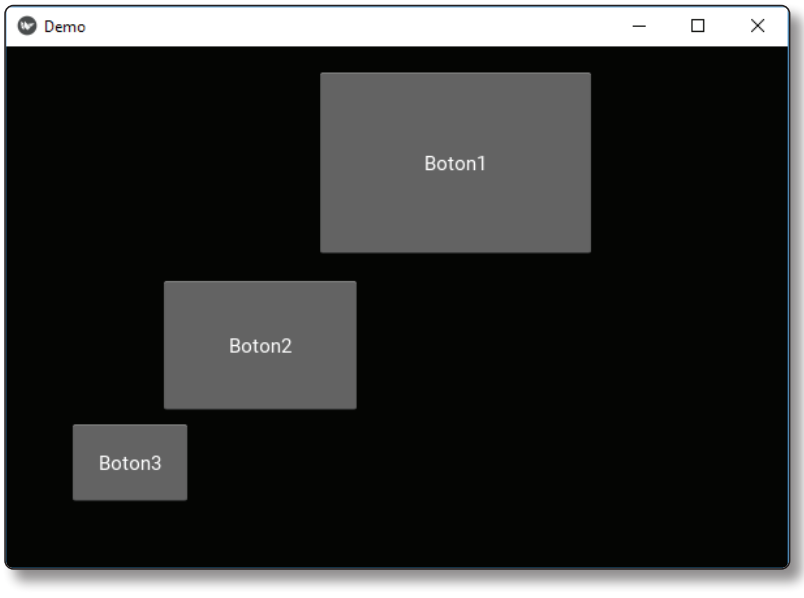
Es muy similar al Floatlayout, solo que este caso cada elemento se acomoda de acuerdo a la posición del layout. Ejemplo:

Stacklayout 1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
Builder.load_string(‘’’
<caja>:
    StackLayout:
        spacing: 5
```

```
        padding: 5
        orientation: 'tb-rl'
        Button:
            text: "boton1"
            size_hint: 0.15, 0.15
        Button:
            text: "boton2"
            size_hint: 0.25, 0.25
        Button:
            text: "boton3"
            size_hint: 0.35, 0.35
'''')
class Caja(BoxLayout):
    pass
class DemoApp(App):
    def build(self):
        return Caja()
if __name__ == '__main__':
    DemoApp().run()
```

Al ejecutarse:



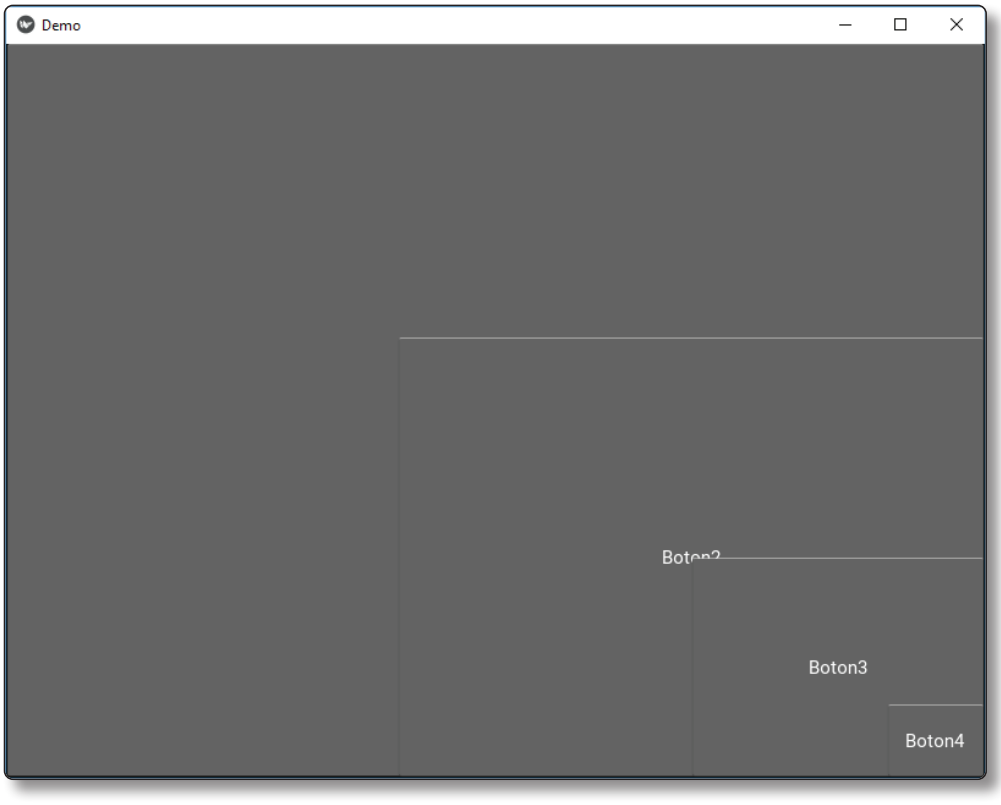
1.9.6 AnchorLayout

Este layout nos permite agregar elementos basándonos en los bordes del mismo. Ejemplo:

AnchorLayout 1.py

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
Builder.load_string('''
<Caja>:
    anchor_x: 'right'
    anchor_y: 'bottom'
    Button:
        id: label1
        size_hint: 1, 1
        text: 'Boton1'
    Button:
        id: label2
        size_hint: 0.6, 0.6
        text: 'Boton2'
    Button:
        id: label3
        size_hint: 0.3, 0.3
        text: 'Boton3'
    Button:
        id: label4
        size_hint: 0.1, 0.1
        text: 'Boton4'
''')
Window.size = (800, 600)
class Caja(AnchorLayout):
    pass
class DemoApp(App):
    def build(self):
        return Caja()
if __name__ == '__main__':
    DemoApp().run()
```

Al ejecutarse:



1.10 PERSONALIZANDO MI ETIQUETA

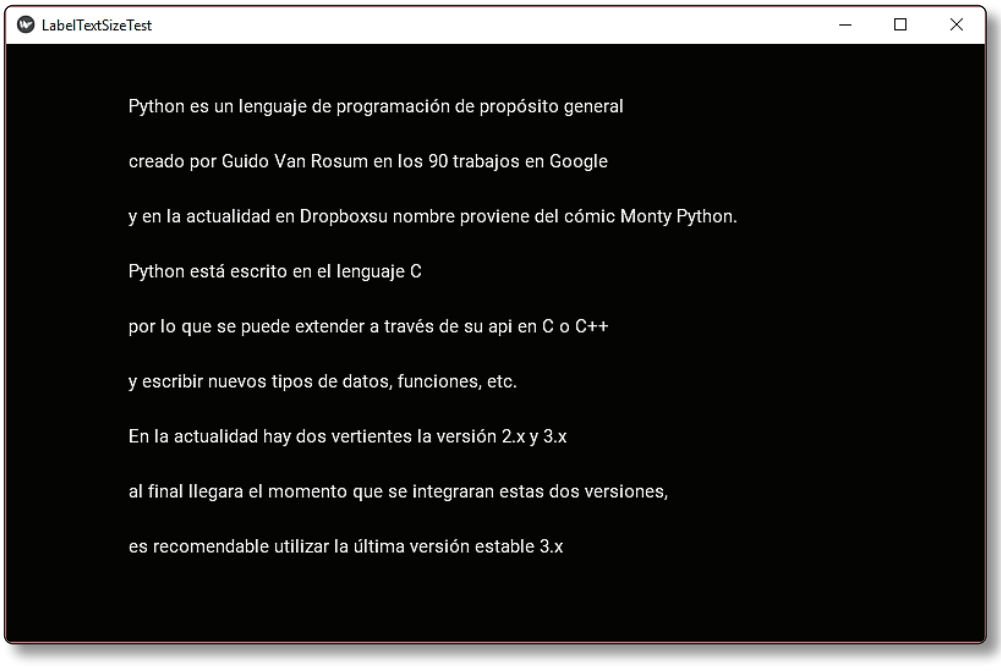
A continuación, mostramos una Aplicación que muestre múltiples líneas de texto:

TextoLargo.py

```
import kivy
kivy.require('1.10.0')
from kivy.app import App
from kivy.uix.button import Label
texto_largo = ("""Python es un lenguaje de programación de propósito general\n"""
"""creado por Guido Van Rosum en los 90 trabajos en Google \ny en la actualidad
```

```
en Dropbox"""
"""su nombre proviene del cómic Monty Python.\n"""
"""Python está escrito en el lenguaje C \n"""
"""por lo que se puede extender a través de su api en C o C++\n"""
"""y escribir nuevos tipos de datos, funciones, etc.\n"""
"""En la actualidad hay dos vertientes la versión 2.x y 3.x\n"""
"""al final llegara el momento que se integraran estas dos versiones,\n"""
"""es recomendable utilizar la última versión estable 3.x""")
class LabelTextSizeTest(App):
    def build(self):
        z = Label(
            text=texto_largo,
            text_size=(600, None),
            line_height=2.5
        )
        return z
if __name__ == '__main__':
    LabelTextSizeTest().run()
```

Al ejecutarse:



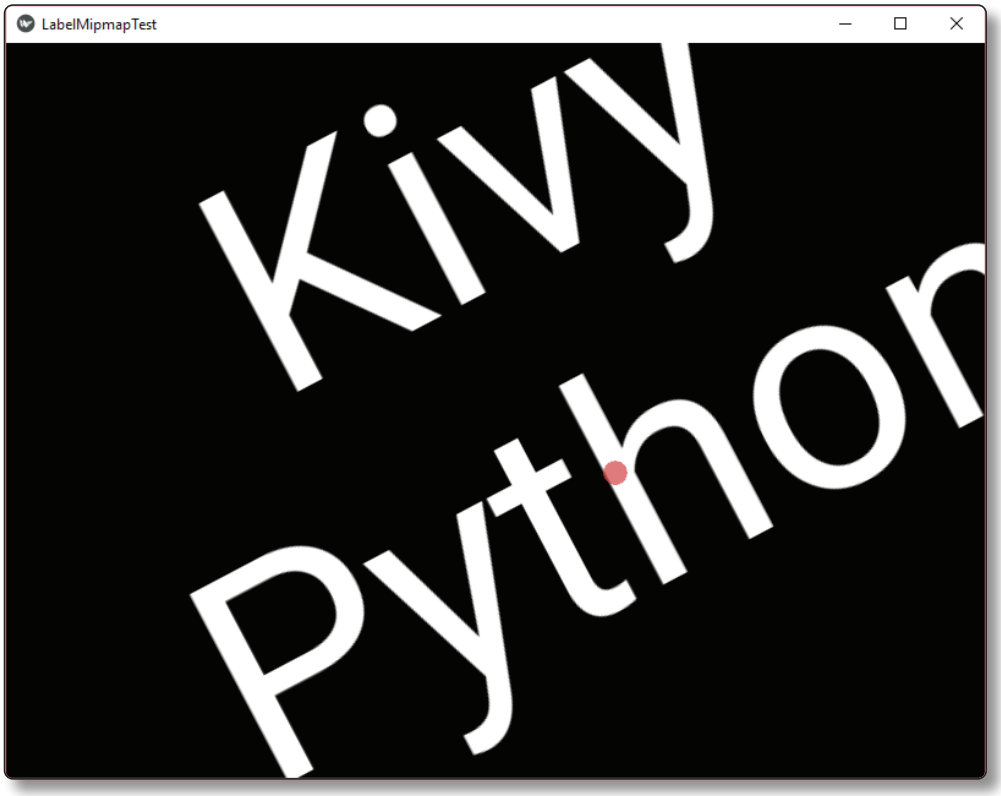
A continuación, mostramos una Aplicación muestre etiquetas (mipmapped label).

TextoLargo.py

```
import kivy
kivy.require('1.10.0')
from kivy.app import App
from kivy.uix.scatter import ScatterPlane
from kivy.uix.label import Label
class LabelMipmapTest(App):
    def build(self):
        s = ScatterPlane(scale=.5)
        l1 = Label(text='Python', font_size=200, pos=(750, 500), mipmap=True)
        l2 = Label(text='Kivy', font_size=200, pos=(750, 728))
        s.add_widget(l1)
        s.add_widget(l2)
        return s

if __name__ == '__main__':
    LabelMipmapTest().run()
```

Al ejecutarse:



NOTAS

Mipmapping es una técnica OpenGL para mejorar la representación de texturas en superficies pequeñas. Sin mipmapping, puede ver pixelación cuando renderiza en superficies pequeñas.

Para que esto suceda, debe especificar mipmap = True cuando crea una textura. Algunos widgets ya le dan la capacidad de crear texturas de mipmapped, como el Label y Image.

A continuación, mostramos una aplicación que muestre cómo cambiar el estilo de texto:

EstiloTexo.py

```
import kivy
kivy.require('1.10.0')
from kivy.app import App
from kivy.lang import Builder
texto_estilo = Builder.load_string('''
Label:
    text:
        ('[color=00563f]Kivy[/color]\\n'
        '[b]biblioteca de Python[/b]\\n'
        '[color=0098c3]Open Source[/color]\\n'
        '[color=088800]para el desarrollo[/color]\\n'
        '[color=8e0d0d]de aplicaciones multitouch[/color]\\n')
    markup: True
    font_size: '44pt'
    halign: 'center'
    valign: 'center'
''')
class LabelEstilo(App):
    def build(self):
        return texto_estilo
if __name__ == '__main__':
    LabelEstilo().run()
```

Al ejecutarse:

A continuación explicaremos el código línea a línea:

Indicamos la versión de Kivy.

```
import kivy
kivy.require('1.10.0')
```

Importamos la clase base de tu App desde app.

```
from kivy.app import App
```

Importamos esta clase para decirle a Kivy que cargue directamente una cadena o un archivo.

```
from kivy.lang import Builder
```

Definiendo el texto con estilo a través de marcadores como:

[b][/b]	**Negrita**
[i][/i]	**Cursiva**
[u][/u]	**Subrayado**
[s][/s]	**Tachado**
[font=<str>][/font]	**Fuentes**
[size=<integer>][/size]	**Tamaño**
[color=#<color>][/color]	**Color**
[ref=<str>][/ref]	**Zona Interactiva**
[anchor=<str>]	**ancla**
[sup][/sup]	**superíndice**

```
texto_estilo = Builder.load_string('''
Label:
    text:
        ('[color=00563f]Kivy[/color]\\n'
        '[b]biblioteca de Python[/b]\\n'
        '[color=0098c3]Open Source[/color]\\n'
        '[color=088800]para el desarrollo[/color]\\n'
        '[color=8e0d0d]de aplicaciones multitouch[/color]\\n')
    markup: True
    font_size: '44pt'
    halign: 'center'
    valign: 'center'
''')
```

Aquí es donde estamos definiendo la clase base de nuestra aplicación Kivy. Solo deberías necesitar cambiar el nombre de su aplicación LabelEstilo en esta línea.

```
class LabelEstilo(App):
```

Función que inicializa y retorna el Widget.

```
    def build(self):
        return texto_estilo
```

La clase LabelEstilo se inicializa y se llama a su método run (). Esto inicializa y comienza nuestro Kivy.

```
if __name__ == "__main__":
    LabelEstilo().run()
```

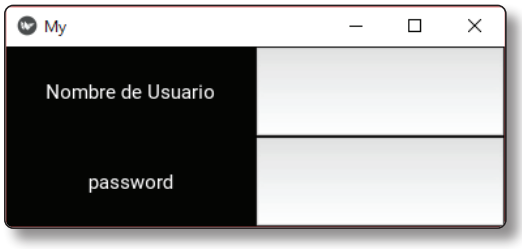
1.10.1 Aplicación Login

A continuación mostramos una aplicación que permita ingresar las credenciales: usuario y contraseña:

Login.py

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
class LoginScreen(GridLayout):
    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.cols = 2
        self.add_widget(Label(text='Nombre de Usuario'))
        self.usuario = TextInput(multiline=False)
        self.add_widget(self.usuario)
        self.add_widget(Label(text='password'))
        self.password = TextInput(password=True, multiline=False)
        self.add_widget(self.password)
class MyApp(App):
    def build(self):
        return LoginScreen()
if __name__ == '__main__':
    MyApp().run()
```

Al ejecutarse:



Importamos la clase base de tu App desde app.

```
from kivy.app import App
```

Importamos Gridlayout.

```
from kivy.uix.gridlayout import GridLayout
```

El uix.label es la sección que contiene los elementos de la interfaz de usuario, como diseños y widgets que se desea mostrar.

```
from kivy.uix.label import Label
```

El uix. textinput es la sección que contiene los componentes de la interfaz de usuario, como por ejemplo un cuadro de texto en el cual el usuario puede introducir datos.

```
from kivy.uix.textinput import TextInput
```

Esta clase se usa como base para nuestro root Widget (LoginScreen).

```
class LoginScreen(GridLayout):
```

En la siguiente sobrecargamos el método __init __ () para agregar widgets y definir su comportamiento.

```
    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
```

Número de columnas.

```
        self.cols = 2
```

Etiqueta Nombre de Usuario.

```
        self.add_widget(Label(text='Nombre de Usuario'))
```

Caja de texto para ingreso de nombre usuario.

```
        self.usuario = TextInput(multiline=False)
        self.add_widget(self.usuario)
```

Etiqueta password.

```
        self.add_widget(Label(text='password'))
```

Caja de Texto para ingreso del password.

```
        self.password = TextInput(password=True, multiline=False)
        self.add_widget(self.password)
```

Aquí es donde estamos definiendo la clase base de nuestra aplicación Kivy. Solo deberías necesitar cambiar el nombre de su aplicación MyApp en esta línea.

```
class MyApp(App):
    def build(self):
```

Función que inicializa y retorna el Widget.

```
        return LoginScreen()
```

La clase MyApp se inicializa y se llama a su método run (). Esto inicializa y comienza nuestro Kivy.

```
if __name__ == '__main__':
    MyApp().run()
```

Ahora añadiremos funcionalidad al login:

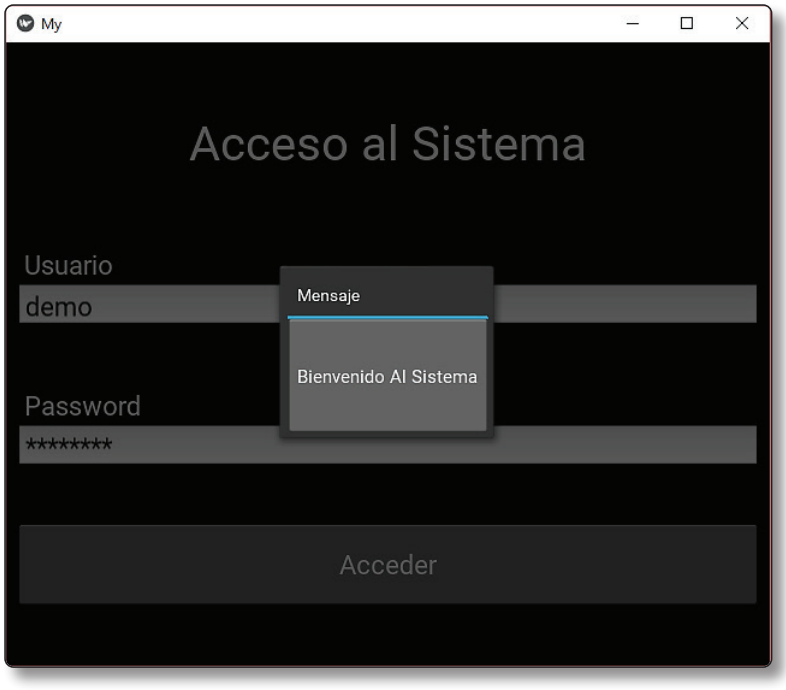
Login.py

```
import kivy
kivy.require('1.10.0')
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.popup import Popup
Builder.load_string("""
<CustomPopup1>:
    size_hint: .3, .3
    auto_dismiss: False
    title: 'Mensaje'
    Button:
        text: 'Bienvenido Al Sistema'
        on_press: root.dismiss()
```

```
<CustomPopup2>:
    size_hint: .3, .3
    auto_dismiss: False
    title: 'Mensaje'
    Button:
        text: 'Credenciales Incorrectos'
        on_press: root.dismiss()
<Login>:
    BoxLayout
        id: login_layout
        orientation: 'vertical'
        padding: [10,50,10,50]
        spacing: 50
        Label:
            text: 'Acceso al Sistema'
            font_size: 40
        BoxLayout:
            orientation: 'vertical'
            Label:
                text: 'Usuario'
                font_size: 22
                halign: 'left'
                text_size: root.width-30, 30
            TextInput:
                id: usuario
                multiline:False
                font_size: 22
        BoxLayout:
            orientation: 'vertical'
            Label:
                text: 'Password'
                halign: 'left'
                font_size: 22
                text_size: root.width-30, 30
            TextInput:
                id: password
                multiline:False
                password:True
                font_size: 22
        Button:
            text: 'Acceder'
            font_size: 22
            on_press: root.loguear(usuario.text, password.text)
"""")
class CustomPopup1(Popup):
    pass
class CustomPopup2(Popup):
    pass
class Login(FloatLayout):
```

```
    def show_popup1(self, b):
        p = CustomPopup1()
        p.open()
    def show_popup2(self, b):
        p = CustomPopup2()
        p.open()
    def loguear(self, loginText, passwordText):
        app = App.get_running_app()
        app.username = loginText
        app.password = passwordText
        if (app.username==”demo” and app.password==”12345678”):
            p = CustomPopup1()
            p.open()
        else:
            p = CustomPopup2()
            p.open()
class MyApp(App):
    def build(self):
        return Login()
if __name__ == ‘__main__’:
    MyApp().run()
```

Al ejecutarse:



1.11 MI PRIMERA CALCULADORA

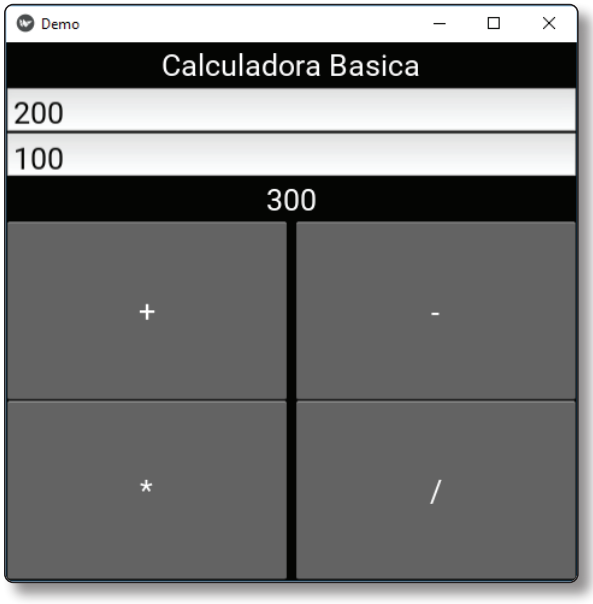
Calculadora.py

```
from kivy.uix.gridlayout import GridLayout
from kivy.app import App
from kivy.lang import Builder
Builder.load_string(‘’’
<CustButton@Button>:
    font_size: 25
    color: 1, 1, 1, 1
<Demo1>:
    numero1: numero1
    numero2: numero2
    resultado : resultado
    cols: 1
    BoxLayout:
        orientation: ‘vertical’
        Label:
```

```
            id: l1
            text: 'Calculadora Basica'
            font_size: 25
        TextInput:
            id: numero1
            multiline:False
            font_size: 25
            text: '20'
        TextInput:
            id: numero2
            multiline:False
            font_size: 25
            text: '10'
        Label:
            id: resultado
            font_size: 25
    BoxLayout:
        spacing: 6
        CustButton:
            size_hint_x: 0.4
            pos_hint: {'x': 0}
            text: '+'
            on_press: root.sumando(*args)
        CustButton:
            size_hint_x: 0.4
            pos_hint: {'y': 0}
            text: '-'
            on_press: root.restando(*args)
    BoxLayout:
        spacing: 6
        CustButton:
            size_hint_x: 0.4
            pos_hint: {'x': 0}
            text: '*'
            on_press: root.multiplicando(*args)
        CustButton:
            size_hint_x: 0.4
            pos_hint: {'y': 0}
            text: '/'
            on_press: root.dividiendo(*args)
''')
class Demo1(GridLayout):
    def sumando(self, instance):
        self.resultado.text = str(int(self.numero1.text) + int
(self.numero2.text))
```

```
    def restando(self, instance):
        self.resultado.text = str(int(self.numero1.text) - int
(self.numero2.text))
    def multiplicando(self, instance):
        self.resultado.text = str(int(self.numero1.text) * int
(self.numero2.text))
    def dividiendo(self, instance):
        try:
            self.resultado.text = str(int(self.numero1.text) / int
(self.numero2.text))
        except ZeroDivisionError:
            self.resultado.text =’division entre zero’
    def reset(self, instance):
        self.numero1.text=””
        self.numero2.text=””
        self.resultado.text=””
class DemoApp(App):
    def build(self):
        return Demo1()
if __name__ == ‘__main__’:
    DemoApp().run()
```

Al ejecutarse:



1.12 USO DE CÁMARA

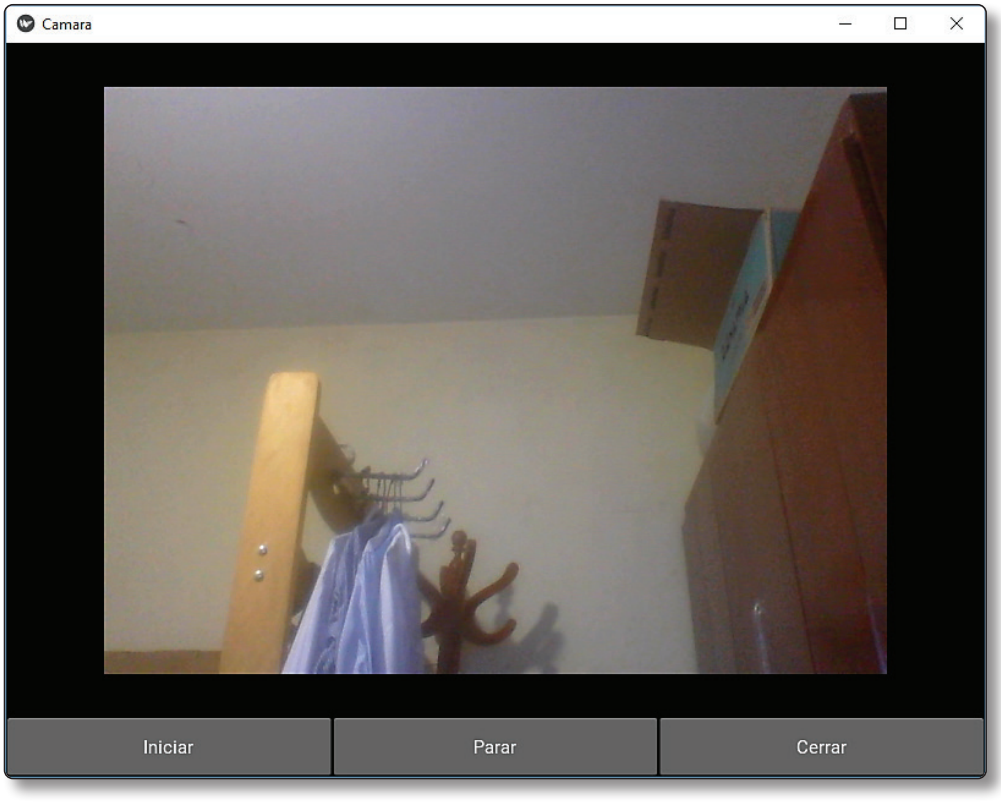
Vamos a desarrollar una aplicación que demuestre el uso de la cámara.

Camara.kv

```
<Caja>:
    orientation: 'vertical'
    Camera:
        id: camera
        resolution: 600, 400
    BoxLayout:
        orientation: 'horizontal'
        size_hint_y: None
        height: '48dp'
        Button:
            text: 'Iniciar'
            on_release: camera.play = True
        Button:
            text: 'Capturar'
            on_press: root.capturar()
        Button:
            text: 'Parar'
            on_release: camera.play = False
        Button:
            text: 'Cerrar'
            on_release:root.stop()
```

Camara.py

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
import time
class Camara(BoxLayout):
    def capture(self):
        camera = self.ids['camera']
        file_name = "capturado"+time.strftime("%Y%m%d_%H%M%S")
        camera.export_to_png(file_name+".png")
class Plantilla(App):
    def build(self):
        return Camara()
if __name__ == '__main__':
    Plantilla().run()
```



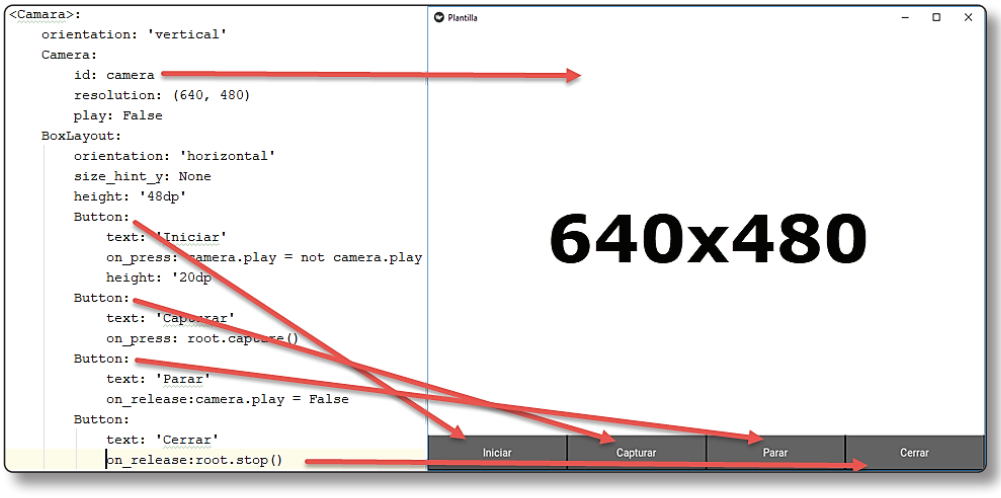
A continuación explicaremos el código línea a línea:

Plantilla.kv

Apreciamos el siguiente código:

```
<Camara>:
    orientation: ‘vertical’
    Camera:
        id: camera
        resolution: (640, 480)
        play: False
    BoxLayout:
        orientation: ‘horizontal’
        size_hint_y: None
        height: ‘48dp’
        Button:
            text: ‘Iniciar’
            on_press: camera.play = not camera.play
            height: ‘20dp’
        Button:
            text: ‘Capturar’
            on_press: root.capture()
        Button:
            text: ‘Parar’
            on_release:camera.play = False
        Button:
            text: ‘Cerrar’
            on_release:root.stop()
```

A continuación realizamos la explicación del código anterior:



camara.kv

Apreciamos el siguiente código:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
import time
class Camara(BoxLayout):
    def capture(self):
        camera = self.ids['camera']
        file_name = "capturado"+time.strftime("%Y%m%d_%H%M%S")
        camera.export_to_png(file_name+".png")
class Plantilla(App):
    def build(self):
        return Camara()
if __name__ == '__main__':
    Plantilla().run()
```

A continuación realizamos la explicación del código anterior:

Manejo de la camara.

```
        camera = self.ids['camera']
```

Nombre del archivo.

```
        file_name = "capturado"+time.strftime("%Y%m%d_%H%M%S")
```

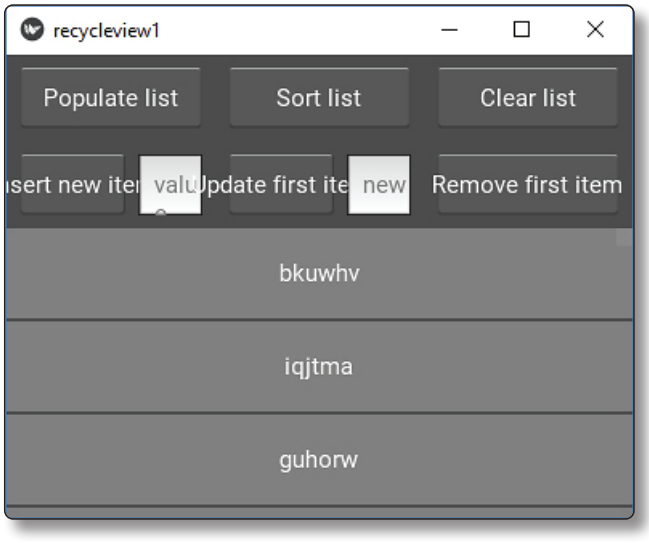
Exportar a un archivo PNG.

```
        camera.export_to_png(file_name+".png")
```

1.13 USO DE RECYCLEVIEW

RecycleView proporciona un modelo flexible para ver secciones seleccionadas de grandes conjuntos de datos. Su objetivo es evitar la disminución del rendimiento que puede ocurrir al cargar grandes cantidades de datos.

La vista se genera mediante el procesamiento de data, esencialmente una lista. Su diseño se basa en el patrón MVC (Model-view-controller).



- Modelo: El modelo está formado por datausted pasa a través de una lista de dicts.
- Vista: La vista se divide en el diseño y las vistas, y se implementa mediante adaters.
- Controlador: El controlador determina la interacción lógica y es implementado por RecycleViewBehavior.

recycleview1.kv

```
<Row@BoxLayout>:
    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            size: self.size
            pos: self.pos
    value: ''
    Label:
        text: root.value
<Test>:
    canvas:
        Color:
            rgba: 0.3, 0.3, 0.3, 1
        Rectangle:
```

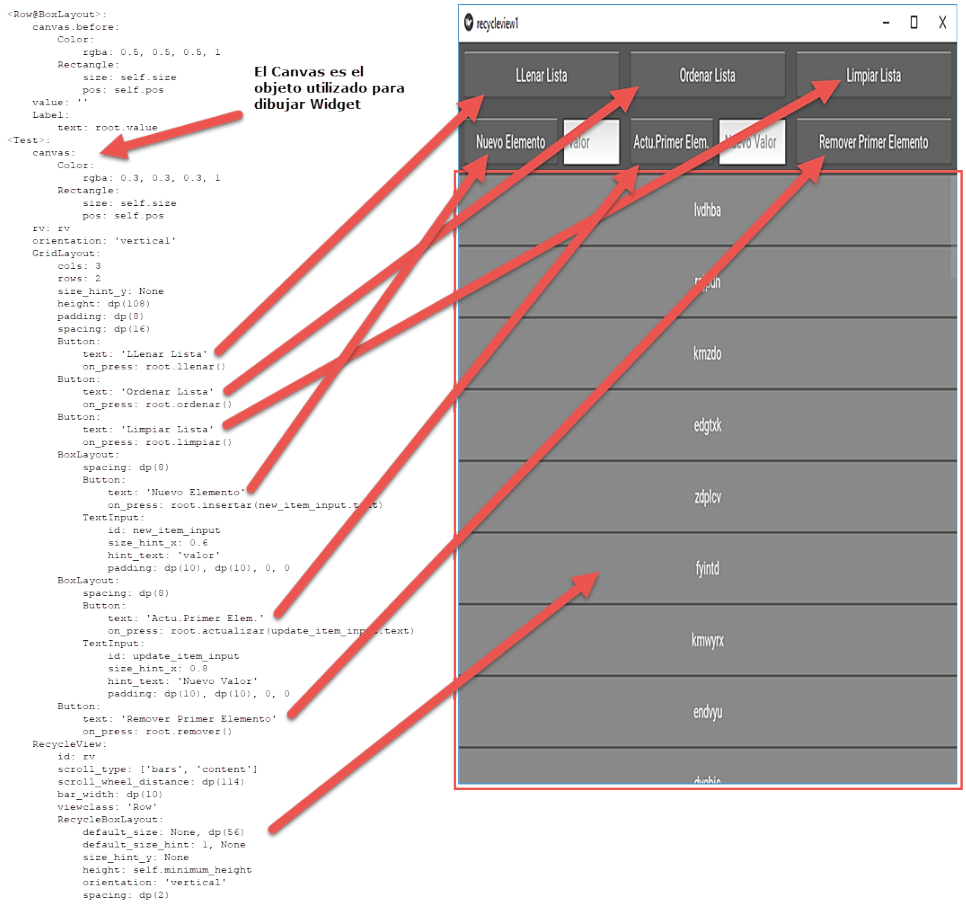
```
            size: self.size
            pos: self.pos
    rv: rv
    orientation: 'vertical'
    GridLayout:
        cols: 3
        rows: 2
        size_hint_y: None
        height: dp(108)
        padding: dp(8)
        spacing: dp(16)
        Button:
            text: 'LLenar Lista'
            on_press: root.llenar()
        Button:
            text: 'Ordenar Lista'
            on_press: root.ordenar()
        Button:
            text: 'Limpiar Lista'
            on_press: root.limpiar()
        BoxLayout:
            spacing: dp(8)
            Button:
                text: 'Nuevo Elemento'
                on_press: root.insertar(new_item_input.text)
            TextInput:
                id: new_item_input
                size_hint_x: 0.6
                hint_text: 'valor'
                padding: dp(10), dp(10), 0, 0
        BoxLayout:
            spacing: dp(8)
            Button:
                text: 'Actu.Primer Elem.'
                on_press: root.actualizar(update_item_input.text)
            TextInput:
                id: update_item_input
                size_hint_x: 0.8
                hint_text: 'Nuevo Valor'
                padding: dp(10), dp(10), 0, 0
        Button:
            text: 'Remover Primer Elemento'
            on_press: root.remover()
    RecycleView:
        id: rv
```

```
        scroll_type: ['bars', 'content']
        scroll_wheel_distance: dp(114)
        bar_width: dp(10)
        viewclass: 'Row'
        RecycleBoxLayout:
            default_size: None, dp(56)
            default_size_hint: 1, None
            size_hint_y: None
            height: self.minimum_height
            orientation: 'vertical'
            spacing: dp(2)
```

recycleview1.py

```
from random import sample
from string import ascii_lowercase
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
class Test(BoxLayout):
    def llenar(self):
        self.rv.data = [{'value': ''.join(sample(ascii_lowercase, 6))}
                        for x in range(50)]
    def ordenar(self):
        self.rv.data = sorted(self.rv.data, key=lambda x: x['value'])
    def limpiar(self):
        self.rv.data = [ ]
    def insertar(self, value):
        self.rv.data.insert(0, {'value': value or 'default value'})
    def actualizar(self, value):
        if self.rv.data:
            self.rv.data[0]['value'] = value or 'default new value'
            self.rv.refresh_from_data()
    def remover(self):
        if self.rv.data:
            self.rv.data.pop(0)
class recycleview1(App):
    def build(self):
        return Test()
if __name__ == '__main__':
    recycleview1().run()
```

A continuación realizamos la explicación del código anterior:



Importamos los módulos a utilizar.

```
from random import sample
from string import ascii_lowercase
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
```

```
class Test(BoxLayout):
```

Función para llenar 50 elementos en el RecycleView.

```
    def llenar(self):
        self.rv.data = [{'value': ''.join(sample(ascii_lowercase, 6))}
                        for x in range(50)]
```

Función para ordenar el RecycleView.

```
    def ordenar(self):
        self.rv.data = sorted(self.rv.data, key=lambda x: x['value'])
```

Función para limpiar el RecycleView.

```
    def limpiar(self):
        self.rv.data = [ ]
```

Función para insertar un elemento en el RecycleView.

```
    def insertar(self, value):
        self.rv.data.insert(0, {'value': value or 'default value'})
```

Función para actualizar un elemento en el RecycleView.

```
    def actualizar(self, value):
        if self.rv.data:
            self.rv.data[0]['value'] = value or 'default new value'
            self.rv.refresh_from_data()
```

Función para remover un elemento en el RecycleView.

```
    def remover(self):
        if self.rv.data:
            self.rv.data.pop(0)
class recycleview1(App):
    def build(self):
        return Test()
if __name__ == '__main__':
    recycleview1().run()
```

1.14 MI PRIMER EDITOR DE TEXTO

Vamos a desarrollar un editor en el cual separaremos el diseño de la lógica de negocio:

Plantilla.kv

```
Root:
    text_input: text_input
    BoxLayout:
        orientation: 'vertical'
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: 'Cargar Fichero'
                on_release: root.cargar()
            Button:
                text: 'Guardar Fichero'
                on_release: root.guardar()
        BoxLayout:
            TextInput:
                id: text_input
                text: ''
            RstDocument:
                text: text_input.text
                show_errors: True
<LoadDialog>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        FileChooserListView:
            id: filechooser
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancelar"
                on_release: root.cancel()
            Button:
                text: "Cargar"
                on_release: root.cargarFichero(filechooser.path,
filechooser.selection)
<SaveDialog>:
    text_input: text_input
    BoxLayout:
        size: root.size
```

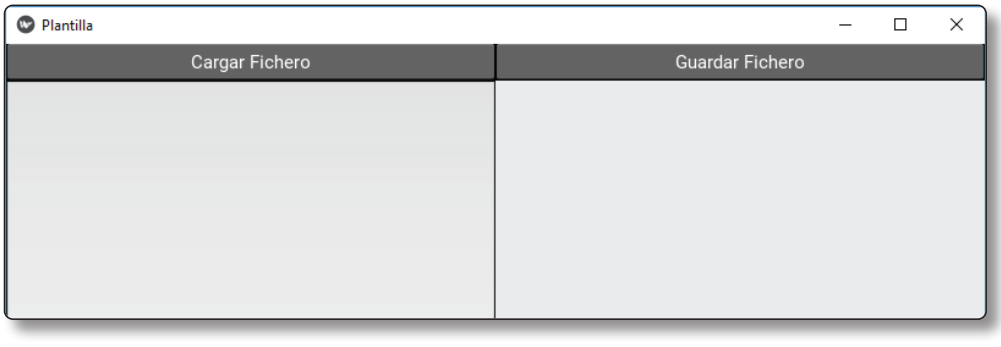
```
        pos: root.pos
        orientation: “vertical”
        FileChooserListView:
            id: filechooser
            on_selection: text_input.text = self.selection and self.selection
[0] or ‘’
        TextInput:
            id: text_input
            size_hint_y: None
            height: 30
            multiline: False
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: “Cancelar”
                on_release: root.cancel()
            Button:
                text: “Guardar”
                on_release: root.guardarFichero(filechooser.path,
text_input.text)
```

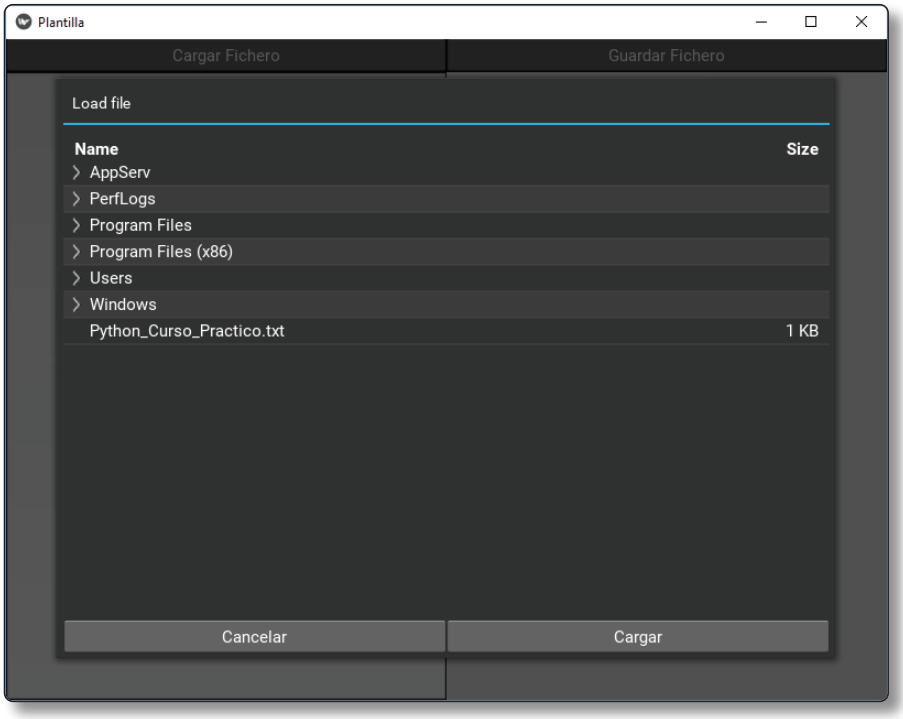
EditorTexto.py

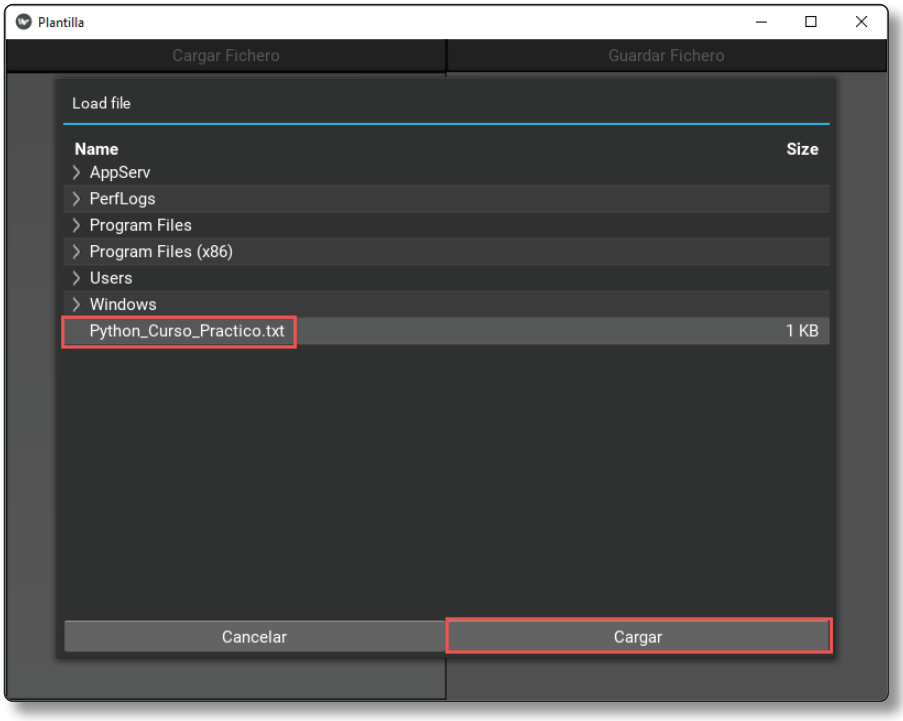
```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.factory import Factory
from kivy.properties import ObjectProperty
from kivy.uix.popup import Popup
import os
class LoadDialog(FloatLayout):
    load = ObjectProperty(None)
    cancel = ObjectProperty(None)
class SaveDialog(FloatLayout):
    save = ObjectProperty(None)
    text_input = ObjectProperty(None)
    cancel = ObjectProperty(None)
class Root(FloatLayout):
    loadfile = ObjectProperty(None)
    savefile = ObjectProperty(None)
    text_input = ObjectProperty(None)
```

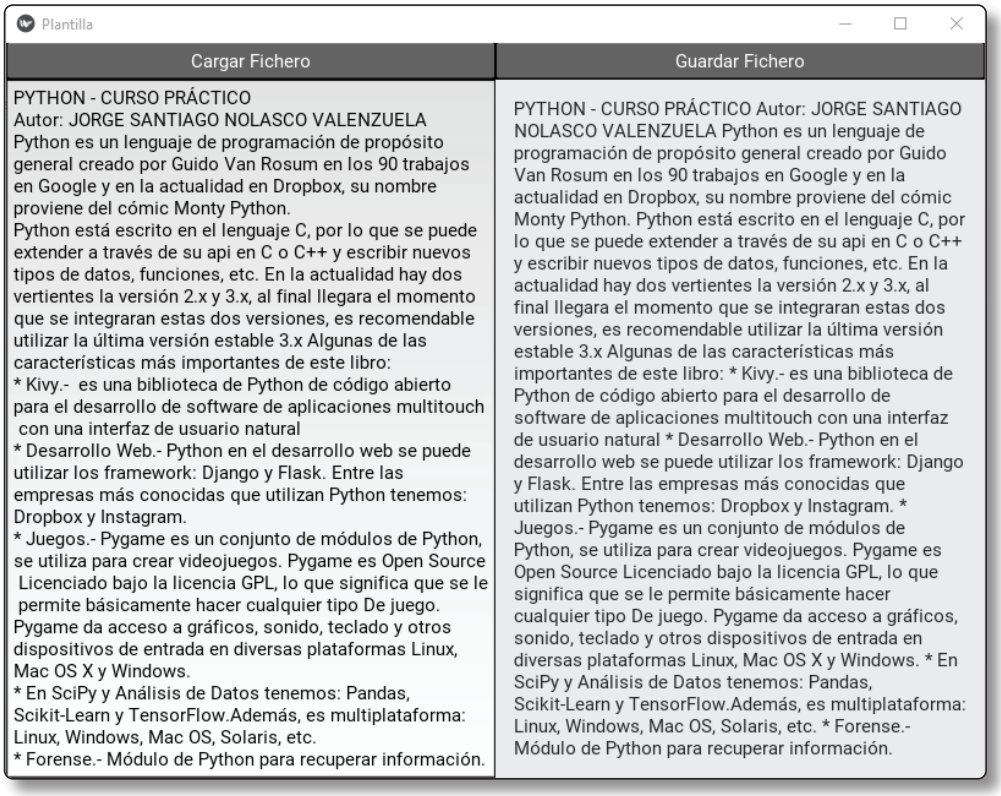
```
    def dismiss_popup(self):
        self._popup.dismiss()
    def show_load(self):
        content = LoadDialog(load=self.load, cancel=self.dismiss_popup)
        self._popup = Popup(title="Load file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()
    def show_save(self):
        content = SaveDialog(save=self.save, cancel=self.dismiss_popup)
        self._popup = Popup(title="Save file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()
    def load(self, path, filename):
        with open(os.path.join(path, filename[0])) as stream:
            self.text_input.text = stream.read()
        self.dismiss_popup()
    def save(self, path, filename):
        with open(os.path.join(path, filename), 'w') as stream:
            stream.write(self.text_input.text)
        self.dismiss_popup()
class Plantilla(App):
    pass
Factory.register('Root', cls=Root)
Factory.register('LoadDialog', cls=LoadDialog)
Factory.register('SaveDialog', cls=SaveDialog)
if __name__ == '__main__':
    Plantilla().run()
```

Al ejecutarse:




Plantilla
Cargar Fichero
Guardar Fichero
Load file
Name
Size
AppServ
PerfLogs
Program Files
Program Files (x86)
Users
Windows
Python_Curso_Practico.txt
1 KB
Cancelar
Cargar


Plantilla
Cargar Fichero
Guardar Fichero
Load file
Name
Size
AppServ
PerfLogs
Program Files
Program Files (x86)
Users
Windows
Python_Curso_Practico.txt
1 KB
Cancelar
Cargar



A continuación explicaremos el código línea a línea:

Plantilla.kv

Hacemos referencia al widget raíz:

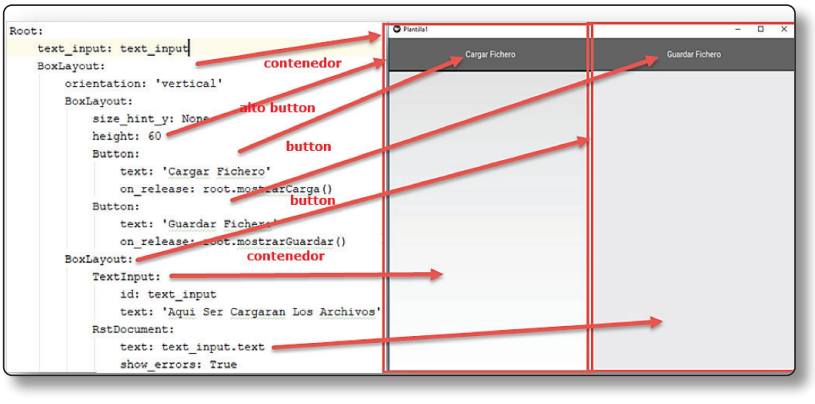
```
Root:text_input: text__y: Noneshow_errors: True
```

Apreciamos el siguiente código:

```
Root:
    text_input: text_input
    BoxLayout:
        orientation: 'vertical'
        BoxLayout:
            size_hint_y: None
            height: 60
            Button:
```

```
                text: 'Cargar Fichero'
                on_release: root.mostrarCarga()
            Button:
                text: 'Guardar Fichero'
                on_release: root.mostrarGuardar()
        BoxLayout:
            TextInput:
                id: text_input
                text: 'Aqui Ser Cargaran Los Archivos'
            RstDocument:
                text: text_input.text
                show_errors: True __y: Noneshow_errors: True
```

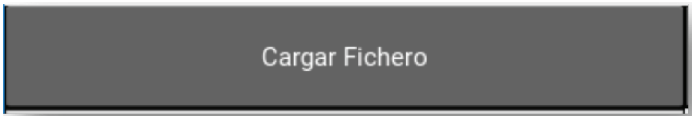
A continuación realizamos la explicación del código anterior:



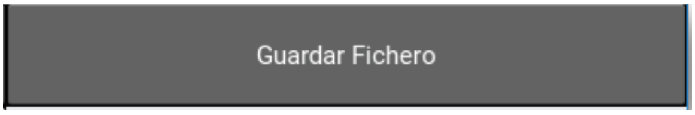
Ahora explicaremos las siguientes líneas:

```
Button:
    text: 'Cargar Fichero'
    on_release: root.cargar()
Button:
    text: 'Guardar Fichero'
    on_release: root.guardar()
```

on_release: root.cargar() .- Se ejecuta la función cargar cuando se presiona el button.

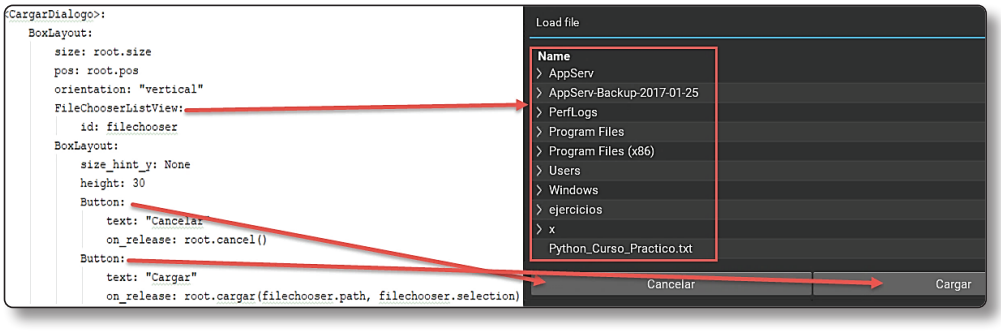


on_release: root.guardar().- Se ejecuta la función cargar cuando se presiona el button.



Apreciamos el siguiente código:

```
<CargarDialogo>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: “vertical”
        FileChooserListView:
            id: filechooser
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: “Cancelar”
                on_release: root.cancel()
            Button:
                text: “Cargar”
                on_release: root.cargar(filechooser.path, filechooser.selection):
```



A continuación realizamos la explicación del código anterior:

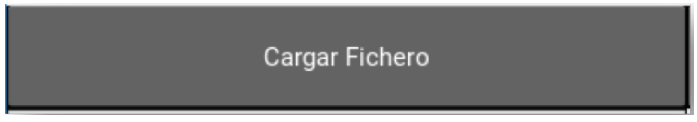
```
            FileChooserListView:
                        id: filechooser
```

El módulo FileChooser proporciona varias clases para describir, visualizar y examinar archivos.

Ahora explicaremos las siguientes líneas:

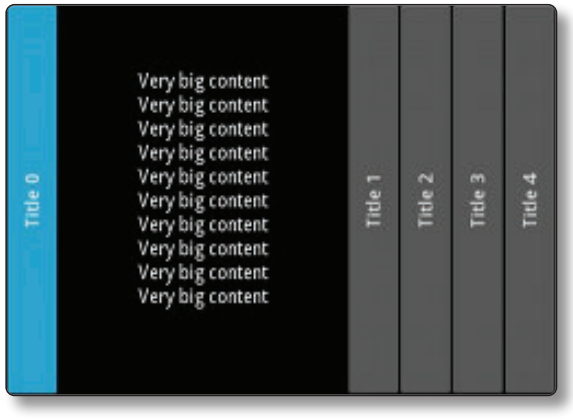
```
Button:
    text: “Cancelar”
    on_release: root.cancel()
Button:
    text: “Cargar”
    on_release: root.cargar(filechooser.path, filechooser.selection)
    on_release: root.cancel()
    on_release: root.cargar(filechooser.path, filechooser.selection)
```

on_release: root.cargar() .- Se ejecuta la función cargar cuando se presiona el button.



on_release: root.guardar().- Se ejecuta la función cargar cuando se presiona el button.

1.15 ACCORDION



El widget Accordion es una forma de menú donde las opciones se apilan vertical u horizontalmente y el elemento en foco (cuando se toca) se abre para mostrar su contenido.

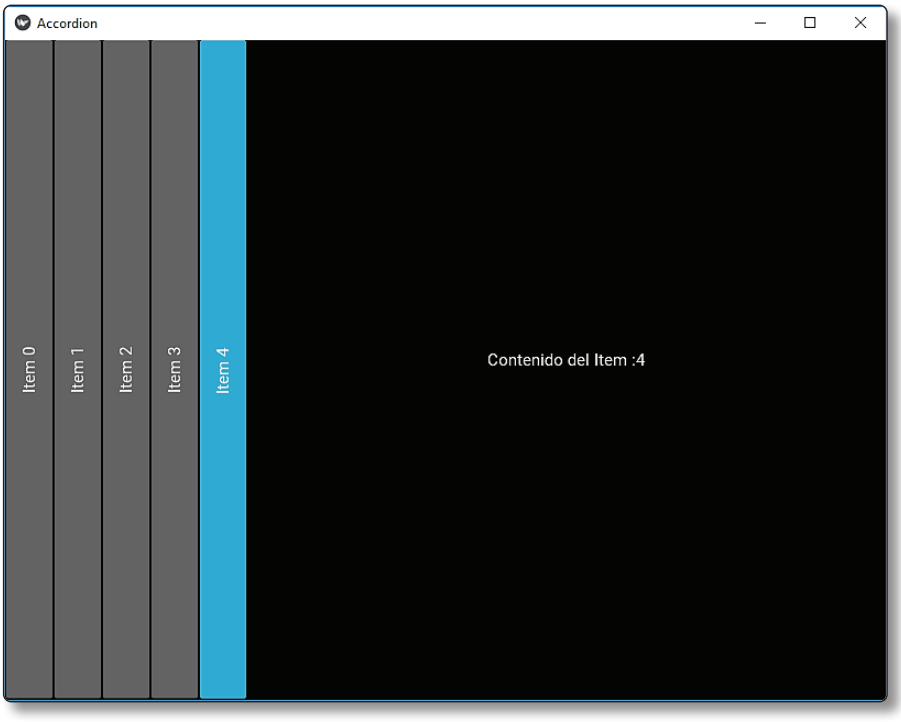
El Accordion debe contener uno o muchos AccordionItem, cada uno de los cuales debe contener widget.

A continuación se muestra un ejemplo de su utilización:

Acordeon.py

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App
class AccordionApp(App):
    def build(self):
        root = Accordion()
        for x in range(5):
            item = AccordionItem(title='Item %d' % x)
            item.add_widget(Label(text='Contenido del Item :'+str(x)+'\n'))
            root.add_widget(item)
        return root
if __name__ == '__main__':
    AccordionApp().run()
```

Al ejecutarse:



1.16 VIDEOS (VIDEO PLAYER)

El widget del reproductor de video se puede usar para reproducir video y dejar que el usuario controle la reproducción/pausa, el volumen y la posición. El widget no se puede personalizar mucho debido al complejo ensamblaje de numerosos widgets básicos.

A continuación, se muestre un ejemplo de su utilización:

Video.py

```
from sys import argv
from os.path import dirname, join
from kivy.app import App
from kivy.uix.videoplayer import VideoPlayer
class Video(App):
    def build(self):
        if len(argv) > 1:
            filename = argv[1]
```

```
        else:
            curdir = dirname(__file__)
            filename = join(curdir, 'kivy.mp4')
        return VideoPlayer(source=filename, state='play')
if __name__ == '__main__':
    Video().run()
```

Al ejecutarse: