



EBook Gratis

APRENDIZAJE

kivy

Free unaffiliated eBook created from
Stack Overflow contributors.

#kivy

Tabla de contenido

Acerca de	1
Capítulo 1: Empezando con Kivy	2
Observaciones.....	2
Examples.....	2
Instalación y configuración.....	2
Windows	2
Caminos.....	4
Simplificarlo.....	4
Toca, agarra y mueve.....	4
Hola mundo en kivy.....	6
Ejemplo emergente simple en Kivy.....	7
RecyclerView.....	7
Diferentes formas de ejecutar una aplicación sencilla e interactuar con widgets.....	8
Con lenguaje .kv.....	10
Capítulo 2: Propiedad	13
Examples.....	13
Diferencia entre propiedades y vinculación.....	13
Capítulo 3: Usando el administrador de pantalla	16
Observaciones.....	16
Importaciones circulares	16
Examples.....	16
Uso simple del administrador de pantalla.....	16
Gestor de pantalla.....	18
Creditos	20

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [kivy](#)

It is an unofficial and free kivy ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official kivy.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Kivy

Observaciones

Kivy es una biblioteca de código abierto de Python para el rápido desarrollo de interfaces de usuario multiplataforma. Las aplicaciones Kivy se pueden desarrollar para Linux, Windows, OS X, Android e iOS usando el mismo código base.

Los gráficos se procesan a través de OpenGL ES 2 en lugar de a través de widgets nativos, lo que lleva a una apariencia bastante uniforme en todos los sistemas operativos.

Desarrollar interfaces en Kivy opcionalmente implica el uso de kvleng, un pequeño lenguaje que admite expresiones similares a python e interoperabilidad de python. El uso de kvleng puede simplificar drásticamente el desarrollo de la interfaz de usuario en comparación con el uso exclusivo de Python.

Kivy es de uso gratuito (actualmente bajo la licencia MIT) y cuenta con respaldo profesional.

Examples

Instalación y configuración

Windows

Hay dos opciones de cómo instalar Kivy:

Primero asegúrese de que las herramientas de Python estén actualizadas.

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

Luego instale las dependencias básicas.

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

Aunque Kivy ya tiene proveedores de audio y video, GStreamer es necesario para cosas más avanzadas.

```
python -m pip install kivy.deps.gstreamer --extra-index-url  
https://kivy.org/downloads/packages/simple/
```

Para hacerlo más simple, `<python>` en el siguiente texto significa una ruta al directorio con el archivo `python.exe`.

1. Rueda

El paquete de la rueda proporciona Kivy compilado, pero con los componentes de fuente de `cython` eliminados, lo que significa que el código del núcleo no se puede compilar de esta manera. El código de Python, sin embargo, es editable.

La versión estable de Kivy está disponible en pypi.

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

La última versión del repositorio oficial está disponible a través de ruedas construidas todas las noches en google drive. Visita el enlace en [documentos que](#) coinciden con tu versión de python. Después de descargar una rueda adecuada, renómbrala para que coincida con el formato de este ejemplo y ejecute el comando.

```
python -m pip install C:\Kivy-1.9.1.dev-cp27-none-win_amd64.whl
```

2. Fuente

Hay más dependencias necesarias para instalar Kivy desde la fuente que para usar las ruedas, pero la instalación es más flexible.

Cree un nuevo archivo en `<python>\Lib\distutils\distutils.cfg` con estas líneas para asegurarse de que se utilizará un compilador adecuado para el código fuente.

```
[build]
compiler = mingw32
```

Entonces se necesita el compilador. Utilice alguno que ya tenga instalado o descargue `mingwpy`. Los archivos importantes, como `gcc.exe`, se encontrarán en `<python>\Scripts`.

```
python -m pip install -i https://pypi.anaconda.org/carlkl/simple mingwpy
```

No olvide configurar las variables de entorno para que Kivy sepa qué proveedores debería usar.

```
set USE_SDL2=1
set USE_GSTREAMER=1
```

Ahora instale las dependencias adicionales requeridas para la compilación.

```
python -m pip install cython kivy.deps.glew_dev kivy.deps.sdl2_dev
python -m pip install kivy.deps.gstreamer_dev --extra-index-url
https://kivy.org/downloads/packages/simple/
```

Consulte la sección `Paths` para asegurarse de que todo esté configurado correctamente e instale Kivy. Elija una de estas opciones:

```
python -m pip install C:\master.zip
python -m pip install https://github.com/kivy/kivy/archive/master.zip
```

Caminos

Kivy necesita un acceso a los binarios de algunas dependencias. Esto significa que las carpetas correctas *deben* estar en la variable `PATH` del entorno.

```
set PATH=<python>\Tools;<python>\Scripts;<python>\share\sdl2\bin;%PATH%
```

De esta manera, el IDE IDE de Python puede incluirse en la ruta con `<python>\Lib\idlelib; .` Luego escriba `idle` en la consola e IDLE estará listo para usar Kivy.

Simplificarlo

Para evitar la configuración repetitiva de las variables de entorno, establezca cada ruta necesaria de [esta manera](#) o haga un archivo por lotes (`.bat`) con estas líneas ubicadas en `<python> :`

```
set PATH=%~dp0;%~dp0Tools;%~dp0Scripts;%~dp0share\sdl2\bin;%~dp0Lib\idlelib;%PATH%
cmd.exe
```

Para ejecutar el proyecto Kivy después de la instalación, ejecute `cmd.exe` o el archivo por lotes y use `python <filename>.py`

instalación en Ubuntu

Para instalar kivy en ubuntu con el ejemplo kivy, abra el terminal y ejecute el siguiente comando

Primero agrega ppa

```
sudo add-apt-repository ppa:kivy-team/kivy
```

Para instalar kivy

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

Para instalar kivy ejemplos.

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

Toca, agarra y mueve.

El siguiente ejemplo crea un lienzo con 2 puntos y 1 línea en medio. Podrás mover el punto y la línea alrededor.

```
from kivy.app import App
from kivy.graphics import Ellipse, Line
from kivy.uix.boxlayout import BoxLayout

class CustomLayout(BoxLayout):
```

```

def __init__(self, **kwargs):
    super(CustomLayout, self).__init__(**kwargs)

    self.canvas_edge = {}
    self.canvas_nodes = {}
    self.nodesize = [25, 25]

    self.grabbed = {}

    #declare a canvas
    with self.canvas.after:
        pass

    self.define_nodes()
    self.canvas.add(self.canvas_nodes[0])
    self.canvas.add(self.canvas_nodes[1])
    self.define_edge()
    self.canvas.add(self.canvas_edge)

def define_nodes(self):
    """define all the node canvas elements as a list"""

    self.canvas_nodes[0] = Ellipse(
        size = self.nodesize,
        pos = [100,100]
    )

    self.canvas_nodes[1] = Ellipse(
        size = self.nodesize,
        pos = [200,200]
    )

def define_edge(self):
    """define an edge canvas elements"""

    self.canvas_edge = Line(
        points = [
            self.canvas_nodes[0].pos[0] + self.nodesize[0] / 2,
            self.canvas_nodes[0].pos[1] + self.nodesize[1] / 2,
            self.canvas_nodes[1].pos[0] + self.nodesize[0] / 2,
            self.canvas_nodes[1].pos[1] + self.nodesize[1] / 2
        ],
        joint = 'round',
        cap = 'round',
        width = 3
    )

def on_touch_down(self, touch):

    for key, value in self.canvas_nodes.items():
        if (value.pos[0] - self.nodesize[0]) <= touch.pos[0] <= (value.pos[0] +
self.nodesize[0]):
            if (value.pos[1] - self.nodesize[1]) <= touch.pos[1] <= (value.pos[1] +
self.nodesize[1]):
                touch.grab(self)
                self.grabbed = self.canvas_nodes[key]
                return True

```

```

def on_touch_move(self, touch):

    if touch.grab_current is self:
        self.grabbed.pos = [touch.pos[0] - self.nodesize[0] / 2, touch.pos[1] -
self.nodesize[1] / 2]
        self.canvas.clear()
        self.canvas.add(self.canvas_nodes[0])
        self.canvas.add(self.canvas_nodes[1])
        self.define_edge()
        self.canvas.add(self.canvas_edge)
    else:
        # it's a normal touch
        pass

def on_touch_up(self, touch):
    if touch.grab_current is self:
        # I receive my grabbed touch, I must ungrab it!
        touch.ungrab(self)
    else:
        # it's a normal touch
        pass

class MainApp(App):

    def build(self):
        root = CustomLayout()
        return root

if __name__ == '__main__':
    MainApp().run()

```

Hola mundo en kivy.

El siguiente código ilustra cómo hacer la aplicación 'hello world' en kivy. Para ejecutar esta aplicación en ios y android, guárdela como main.py y use buildozer.

```

from kivy.app import App
from kivy.uix.label import Label
from kivy.lang import Builder

Builder.load_string('''
<SimpleLabel>:
    text: 'Hello World'
''')

class SimpleLabel(Label):
    pass

class SampleApp(App):
    def build(self):
        return SimpleLabel()

if __name__ == "__main__":
    SampleApp().run()

```


Ejemplo emergente simple en Kivy.

El siguiente código ilustra cómo hacer una ventana emergente simple con Kivy.

```
from kivy.app import App
from kivy.uix.popup import Popup
from kivy.lang import Builder
from kivy.uix.button import Button

Builder.load_string('''
<SimpleButton>:
    on_press: self.fire_popup()
<SimplePopup>:
    id:pop
    size_hint: .4, .4
    auto_dismiss: False
    title: 'Hello world!!'
    Button:
        text: 'Click here to dismiss'
        on_press: pop.dismiss()
''')

class SimplePopup(Popup):
    pass

class SimpleButton(Button):
    text = "Fire Popup !"
    def fire_popup(self):
        pops=SimplePopup()
        pops.open()

class SampleApp(App):
    def build(self):
        return SimpleButton()

SampleApp().run()
```

RecycleView

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.button import Button

items = [
    {"color":(1, 1, 1, 1), "font_size": "20sp", "text": "white", "input_data":
["some","random","data"]},
    {"color":(.5,1, 1, 1), "font_size": "30sp", "text": "lightblue", "input_data": [1,6,3]},
    {"color":(.5,.5,1, 1), "font_size": "40sp", "text": "blue", "input_data": [64,16,9]},
    {"color":(.5,.5,.5,1), "font_size": "70sp", "text": "gray", "input_data":
[8766,13,6]},
    {"color":(1,.5,.5, 1), "font_size": "60sp", "text": "orange", "input_data": [9,4,6]},
    {"color":(1, 1,.5, 1), "font_size": "50sp", "text": "yellow", "input_data":
[852,958,123]}
]
```

```

class MyButton(Button):

    def print_data(self, data):
        print(data)

KV = '''

<MyButton>:
    on_release:
        root.print_data(self.input_data)

RecycleView:
    data: []
    viewclass: 'MyButton'
    RecycleBoxLayout:
        default_size_hint: 1, None
        orientation: 'vertical'

'''

class Test(App):
    def build(self):
        root = Builder.load_string(KV)
        root.data = [item for item in items]
        return root

Test().run()

```

Diferentes formas de ejecutar una aplicación sencilla e interactuar con widgets.

La mayoría de las aplicaciones kivy comienzan con esta estructura:

```

from kivy.app import App

class TutorialApp(App):
    def build(self):
        return

TutorialApp().run()

```

Hay varias maneras de ir desde aquí:

Todos los códigos a continuación (excepto los ejemplos 1 y 3) tienen el mismo widget y características similares, pero muestran una forma diferente de crear la aplicación.

Ejemplo 1: devolver un solo widget (aplicación Hello World simple)

```

from kivy.app import App
from kivy.uix.button import Button
class TutorialApp(App):
    def build(self):
        return Button(text="Hello World!")
TutorialApp().run()

```

Ejemplo 2: devolver varios widgets + el botón imprime el texto de la etiqueta

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button

class TutorialApp(App):
    def build(self):
        mylayout = BoxLayout(orientation="vertical")
        mylabel = Label(text= "My App")
        mybutton =Button(text="Click me!")
        mylayout.add_widget(mylabel)
        mybutton.bind(on_press= lambda a:print(mylabel.text))
        mylayout.add_widget(mybutton)
        return mylayout
TutorialApp().run()
```

Ejemplo 3: usando una clase (widget único) + el botón imprime "Mi botón"

```
from kivy.app import App
from kivy.uix.button import Button

class Mybutton(Button):
    text="Click me!"
    on_press =lambda a : print("My Button")

class TutorialApp(App):
    def build(self):
        return Mybutton()
TutorialApp().run()
```

Ejemplo 4: es lo mismo que ej. 2 pero muestra como usar una clase

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button

class MyLayout(BoxLayout):
    #You don't need to understand these 2 lines to make it work!
    def __init__(self, **kwargs):
        super(MyLayout, self).__init__(**kwargs)

        self.orientation="vertical"
        mylabel = Label(text= "My App")
        self.add_widget(mylabel)
        mybutton =Button(text="Click me!")
        mybutton.bind(on_press= lambda a:print(mylabel.text))
        self.add_widget(mybutton)

class TutorialApp(App):
    def build(self):
        return MyLayout()
TutorialApp().run()
```

Con lenguaje .kv

Ejemplo 5: lo mismo pero que muestra cómo usar el lenguaje kv *dentro de python*

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
# BoxLayout: it's in the python part, so you need to import it

from kivy.lang import Builder
Builder.load_string("""
<MyLayout>
    orientation:"vertical"
    Label: # it's in the kv part, so no need to import it
        id:mylabel
        text:"My App"
    Button:
        text: "Click me!"
        on_press: print(mylabel.text)
""")
class MyLayout(BoxLayout):
    pass
class TutorialApp(App):
    def build(self):
        return MyLayout()
TutorialApp().run()
```

**** Ejemplo 6: lo mismo con la parte kv en un archivo Tutorial.kv ****

En .py:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout

class MyLayout(BoxLayout):
    pass
class TutorialApp(App):
#the kv file name will be Tutorial (name is before the "App")
    def build(self):
        return MyLayout()
TutorialApp().run()
```

En Tutorial.kv:

```
<MyLayout> # no need to import stuff in kv!
    orientation:"vertical"
    Label:
        id:mylabel
        text:"My App"
    Button:
        text: "Click me!"
        on_press: print(mylabel.text)
```

**** Ejemplo 7: enlace a un archivo kv específico + una definición en python que recibe el label.text ****

En .py:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout

class MyLayout(BoxLayout):
    def printMe(self_xx, yy):
        print(yy)
class TutorialApp(App):
    def build(self):
        self.load_kv('myapp.kv')
        return MyLayout()
TutorialApp().run()
```

En myapp.kv: orientación: "vertical" Etiqueta: id: mylabel text: "My App" Button: text: "Click me!"
on_press: root.printMe (mylabel.text)

Ejemplo 8: el botón imprime el texto de la etiqueta (con una definición en python usando los ids (las "ID"))

Darse cuenta de:

- `self_xx` del ejemplo 7 es reemplazado por `self`

En .py:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout

class MyLayout(BoxLayout):
    def printMe(self):
        print(self.ids.mylabel.text)
class TutorialApp(App):
    def build(self):
        self.load_kv('myapp.kv')
        return MyLayout()
TutorialApp().run()
```

En myapp.kv:

```
<MyLayout>
orientation:"vertical"
Label:
    id:mylabel
    text:"My App"
Button:
    text: "Click me!"
    on_press: root.printMe()
```

Ejemplo 9: el botón imprime el texto de la etiqueta (con una definición en python usando StringProperty)

En .py:

```

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import StringProperty
class MyLayout(BoxLayout):
    stringProperty_mylabel= StringProperty("My App")
    def printMe(self):
        print(self.stringProperty_mylabel)

class TutorialApp(App):
    def build(self):
        return MyLayout()
TutorialApp().run()

```

En Tutorial.kv:

```

<MyLayout>
orientation:"vertical"
Label:
    id:mylabel
    text:root.stringProperty_mylabel
Button:
    text: "Click me!"
    on_press: root.printMe()

```

Ejemplo 10: el botón imprime el texto de la etiqueta (con una definición en python usando ObjectProperty)

En .py:

```

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import ObjectProperty
class MyLayout(BoxLayout):
    objectProperty_mylabel= ObjectProperty(None)
    def printMe(self):
        print(self.objectProperty_mylabel.text)

class TutorialApp(App):
    def build(self):
        return MyLayout()
TutorialApp().run()

```

En Tutorial.kv:

```

<MyLayout>
orientation:"vertical"
objectProperty_mylabel:mylabel
Label:
    id:mylabel
    text:"My App"
Button:
    text: "Click me!"
    on_press: root.printMe()

```

Lea Empezando con Kivy en línea: <https://riptutorial.com/es/kivy/topic/2101/empezando-con-kivy>

Capítulo 2: Propiedad

Examples

Diferencia entre propiedades y vinculación.

En breve:

- Las propiedades facilitan la transferencia de actualizaciones desde el lado de Python a la interfaz de usuario
- El enlace pasa los cambios que ocurrieron en la interfaz de usuario al lado de Python.

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder
from kivy.properties import StringProperty
from kivy.properties import ObjectProperty
from kivy.uix.textinput import TextInput
from kivy.event import EventDispatcher
Builder.load_string("""
<CustLab1@Label>
    size_hint:0.3,1
<CustLab2@Label>
    text: "Result"
    size_hint: 0.5,1
<CustButton@Button>
    text: "+1"
    size_hint: 0.1,1
<CustTextInput@TextInput>:
    multiline: False
    size_hint:0.1,1

<Tuto_Property>:
    orientation: "vertical"
    padding:10,10
    spacing: 10
    Label:
        text: "Press the 3 button (+1) several times and then modify the number in the
TextInput.The first counter (with StringProperty but no binding) doesn't take into account the
change that happened in the app, but the second one does.String Property makes it easy to pass
the update from the python side to the user interface, binding pass the changes that happened
on the user interface to the python side. "
        text_size: self.size
        padding: 20,20

    Property_no_Binding:
    Property_with_Binding:
    Simple:

<Property_no_Binding>:
    spacing: 10
    label_ObjectProperty: result
    CustLab1:
        text: "With Property but no Binding"
    CustButton:
```

```

        on_press: root.counter_textInput_StringProperty()
CustTextInput:
    id:textinput_id
    text: root.textInput_StringProperty
CustLab2:
    id: result

<Property_with_Binding>:
    spacing: 10
    label_ObjectProperty: result
CustLab1:
    text: "With Property and Binding"
CustButton:
    on_press: root.counter_textInput_StringProperty()
CustTextInput:
    id:textinput_id
    text: root.textInput_StringProperty
    on_text: root.textInput_StringProperty = self.text    ## this is the binding
CustLab2:
    id: result

<Simple>
    spacing: 10
CustLab1:
    text: "Without Property"
CustButton:
    on_press: root.simple(textinput_id, result)
CustTextInput:
    id:textinput_id
    text: "0"
CustLab2:
    id: result

""")

class Property_no_Binding(BoxLayout):
    textInput_StringProperty= StringProperty("0")
    label_ObjectProperty = ObjectProperty(None)
    def counter_textInput_StringProperty(self):
        self.label_ObjectProperty.text= ("Before the counter was updated:\n\n
textInput_id.text:" + self.ids.textinput_id.text + "\n\n textInput_StringProperty:" +
self.textInput_StringProperty)
        self.textInput_StringProperty =str(int(self.textInput_StringProperty)+1)

class Property_with_Binding(BoxLayout):
    textInput_StringProperty= StringProperty("0")
    label_ObjectProperty = ObjectProperty(None)
    def counter_textInput_StringProperty(self):
        self.label_ObjectProperty.text= ("Before the counter was updated:\n\n
textInput_id.text:" + self.ids.textinput_id.text + "\n\n textInput_StringProperty:" +
self.textInput_StringProperty)
        self.textInput_StringProperty =str(int(self.textInput_StringProperty)+1)
    pass

class Simple(BoxLayout):
    def simple(self,textinput_id, result):
        result.text = ("Before the counter was updated:\n\nIn the TextInput:" +
textInput_id.text)
        textinput_id.text = str(int(textinput_id.text) + 1)

```



```

        pass
class Tuto_Property(BoxLayout):

    # def __init__(self, **kwargs):
    #     super(All, self).__init__(**kwargs)
    #     app=App.get_running_app()
    #     self.objproper_number.bind(text=lambda *a: self.change(app))
    #     print(self.parent)

    # def counter(self,app):
    #     print("Stringproperty:",app.numbertext)
    #     print("ObjectProperty:",self.objproper_number.text)
    #     print("text:",self.ids.number.text,"\n")
    #     app.numbertext=str(int(app.numbertext)+1)

    # def change(self, app):
    #     app.numbertext=self.objproper_number.text
    pass
class MyApp(App):
    numbertext = StringProperty("0")
    def build(self):
        return Tuto_Property()

MyApp().run()

```

Lea Propiedad en línea: <https://riptutorial.com/es/kivy/topic/9904/propiedad>

Capítulo 3: Usando el administrador de pantalla

Observaciones

Importaciones circulares

Este es un gran problema en Kivy, Python y muchos lenguajes de programación

Cuando dos recursos requieren un recurso, es normal colocar este recurso en el archivo que más lo utilizará. Pero si esto sucede con dos recursos, y terminan en archivos opuestos, la importación de ambos en Python resultará en una importación circular.

Python importará el primer archivo, pero este archivo importará el segundo. En el segundo, esto importa el primer archivo, que a su vez importa el segundo y así sucesivamente. Python lanza el error `ImportError : cannot import name <classname>`

Esto se puede resolver utilizando un tercer archivo e importando este tercer archivo a los dos primeros. Esto es `resources.py` en el segundo ejemplo.

Examples

Uso simple del administrador de pantalla

```
# A line used mostly as the first one, imports App class
# that is used to get a window and launch the application
from kivy.app import App

# Casual Kivy widgets that reside in kivy.uix
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.screenmanager import SlideTransition

# Inherit Screen class and make it look like
# a simple page with navigation

class CustomScreen(Screen):

    # It's necessary to initialize a widget the class inherits
    # from to access its methods such as 'add_widget' with 'super()'

    def __init__(self, **kwargs):
        # Py2/Py3 note: although in Py3 'super()' is simplified
        # it's a good practice to use Py2 syntax, so that the
        # code is compatible in both versions
        super(CustomScreen, self).__init__(**kwargs)
```

```

# Put a layout in the Screen which will take
# Screen's size and pos.

# The 'orientation' represents a direction
# in which the widgets are added into the
# BoxLayout - 'horizontal' is the default
layout = BoxLayout(orientation='vertical')

# Add a Label with the name of Screen
# and set its size to 50px
layout.add_widget(Label(text=self.name, font_size=50))

# Add another layout to handle the navigation
# and set the height of navigation to 20%
# of the CustomScreen
navig = BoxLayout(size_hint_y=0.2)

# Create buttons with a custom text
prev = Button(text='Previous')
next = Button(text='Next')

# Bind to 'on_release' events of Buttons
prev.bind(on_release=self.switch_prev)
next.bind(on_release=self.switch_next)

# Add buttons to navigation
# and the navigation to layout
navig.add_widget(prev)
navig.add_widget(next)
layout.add_widget(navig)

# And add the layout to the Screen
self.add_widget(layout)

# *args is used to catch arguments that are returned
# when 'on_release' event is dispatched

def switch_prev(self, *args):
    # 'self.manager' holds a reference to ScreenManager object
    # and 'ScreenManager.current' is a name of a visible Screen
    # Methods 'ScreenManager.previous()' and 'ScreenManager.next()'
    # return a string of a previous/next Screen's name
    self.manager.transition = SlideTransition(direction="right")
    self.manager.current = self.manager.previous()

def switch_next(self, *args):
    self.manager.transition = SlideTransition(direction="right")
    self.manager.current = self.manager.next()

class ScreenManagerApp(App):

    # 'build' is a method of App used in the framework it's
    # expected that the method returns an object of a Kivy widget

    def build(self):
        # Get an object of some widget that will be the core
        # of the application - in this case ScreenManager
        root = ScreenManager()

```

```

# Add 4 CustomScreens with name 'Screen <order>'
for x in range(4):
    root.add_widget(CustomScreen(name='Screen %d' % x))

# Return the object
return root

# This is only a protection, so that if the file
# is imported it won't try to launch another App

if __name__ == '__main__':
    # And run the App with its method 'run'
    ScreenManagerApp().run()

```

Gestor de pantalla

En el siguiente ejemplo hay 2 pantallas: SettingsScreen y MenuScreen

Usando el primer botón, en la pantalla actual cambiará su pantalla a la otra pantalla.

Aquí está el código:

```

from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen

# Create both screens. Please note the root.manager.current: this is how
# you can control the ScreenManager from kv. Each screen has by default a
# property manager that gives you the instance of the ScreenManager used.
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'First Button on Menu'
            on_press: root.manager.current = 'settings'
        Button:
            text: 'Second Button on Menu'

<SettingsScreen>:
    BoxLayout:
        Button:
            text: 'First Button on Settings'
            on_press: root.manager.current = 'menu'
        Button:
            text: 'Second Button on Settings'

""")

# Declare both screens
class MenuScreen(Screen):
    pass

class SettingsScreen(Screen):
    pass

# Create the screen manager
sm = ScreenManager()

```

```
sm.add_widget(MenuScreen(name='menu'))
sm.add_widget(SettingsScreen(name='settings'))

class TestApp(App):

    def build(self):
        return sm

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

Lea Usando el administrador de pantalla en línea:

<https://riptutorial.com/es/kivy/topic/6097/usando-el-administrador-de-pantalla>

Creditos

S. No	Capítulos	Contributors
1	Empezando con Kivy	Community , Daniel Engel , EL3PHANTEN , Enora , Fermi paradox , JinSnow , Kallz , KeyWeeUsr , phunsukwangdu , picibucor , user2314737 , Will
2	Propiedad	Enora , YOSHI
3	Usando el administrador de pantalla	KeyWeeUsr , M Ganesh , OllieNye , picibucor