PROJECT OVERWIEW

Kivy Application

Pushpendra Tiwari
SRM University

2.My project : How to create Kivy application.

2.1 Kivy

2.1.1 What is Kivy ?

Kivy - Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps. Kivy is the main framework developed by the Kivy organization, alongside Python for Android, Kivy iOS, and several other libraries meant to be used on all platforms.

2.1.2 Specifications:

The framework contains all the elements for building an application such as:

- extensive input support for mouse, keyboard, TUIO, and OS-specific multitouch events,
- a graphic library using only OpenGL ES 2, and based on Vertex Buffer Object and shaders,
- a wide range of Widgets that support multitouch,
- an intermediate language (Kv)used to easily design custom Widgets.

Kivy provides the following properties:

`NumericProperty`, `StringProperty`, `ListProperty`, `ObjectProperty`, `BooleanProperty`, `BoundedNumericProperty`, `OptionProperty`, `ReferenceListProperty`, `AliasProperty`, `DictProperty`,

2.1.3 Characteristic of the market:

• Competitors

-The principal competitor is Android.
-Iphone OS
-Palm OS devices on PDA.
-Blackberry: which team the same name smartphones
-Windows Mobile: which team smartphones and PDAs.
-Sybian: Current Market Leader

• Market share

As Kivy is very new to app development arena therefore, its not that popular among developers. But its Multiplatform and Highly rich libraries are going to attract large number of geeky minds in upcoming future.

2.1.4 Why Kivy is better ?

• Applications

-Cross platform

Kivy runs on Linux, Windows, OS X, Android and iOS. You can run the same code on all supported platforms. It can use natively most inputs, protocols and devices including WM_Touch, WM_Pen, Mac OS X Trackpad and Magic Mouse, Mtdev, Linux Kernel HID, TUIO. A multi-touch mouse simulator is included.

-Business Friendly

Kivy is 100% free to use, under an MIT license (starting from 1.7.2) and LGPL 3 for the previous versions. The toolkit is professionally developed, backed and used. You can use it in a commercial product. The framework is stable and has a well documented API, plus a programming guide to help you get started.

-GPU Accelerated

The graphics engine is built over OpenGL ES 2, using a modern and fast graphics pipeline. The toolkit comes with more than 20 widgets, all highly extensible. Many parts are written in C using Cython, and tested with regression tests.

2.2 First Project : Animalia

-Kivy Basics

-Installation of the Kivy environment

Kivy depends on many Python libraries, such as pygame, gstreamer, PIL, Cairo, and more. They are not all required, but depending on the platform you’re working on, they can be a pain to install. For Windows and MacOS X, we provide a port-able package that you can just unzip and use.

2.2.1 Installation on Linux

Using software packages

For installing distribution relative packages .deb/.rpm/...

Ubuntu / Kubuntu / Xubuntu / Lubuntu (Saucy and above)

1. Add one of the PPAs as you prefer

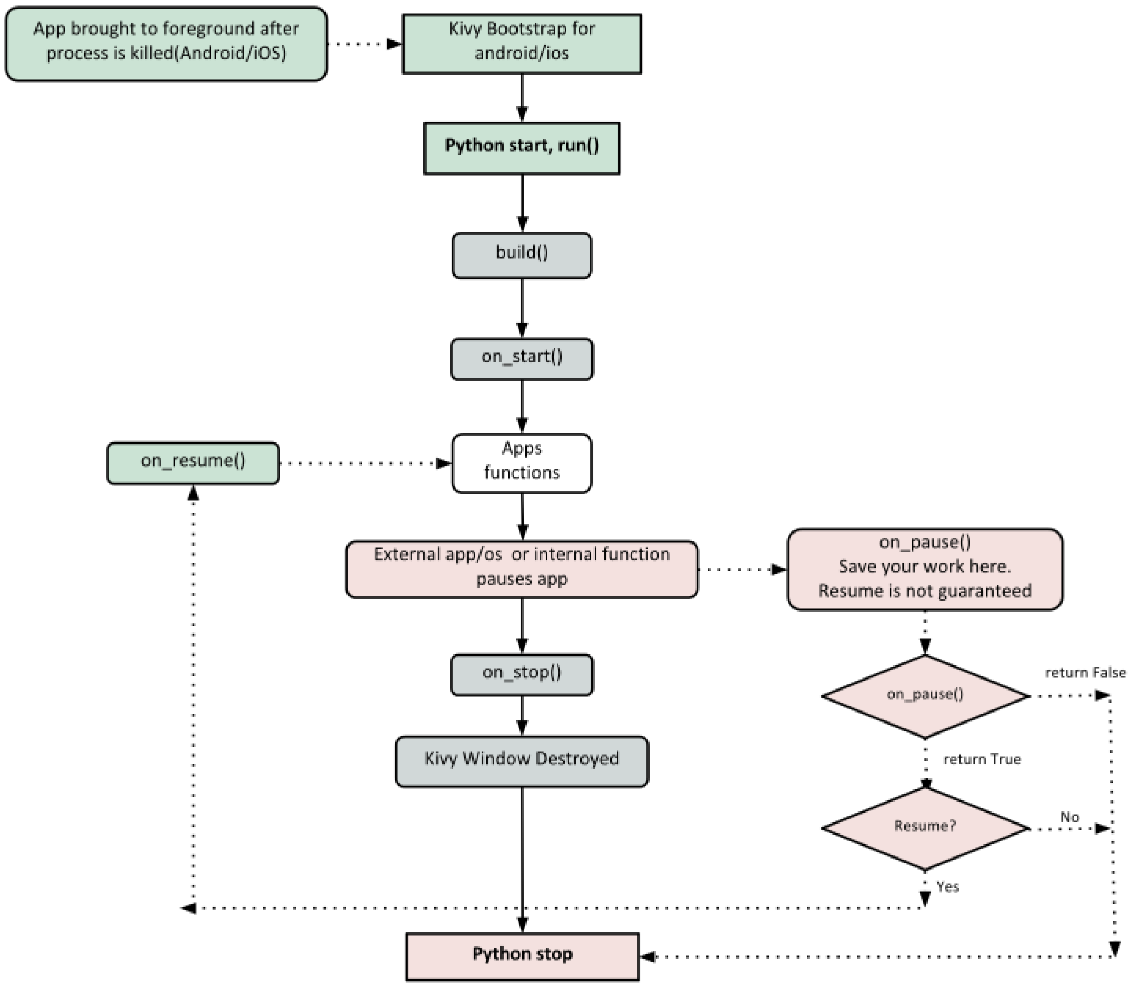
stable builds:	$ sudo add-apt-repository ppa:kivy-team/kivy
nightly builds:	$ sudo add-apt-repository ppa:kivy-team/kivy-daily

2. Update your packagelist using your package manager
3. Install Kivy

Python2 - python-kivy:
$ sudo apt-get install python-kivy
Python3 - python3-kivy:
$ sudo apt-get install python3-kivy
optionally the examples - kivy-examples:
$ sudo apt-get install kivy-examples

2.2.2 Kivy App Life Cycle

Kivy app life cycle


App brought to foreground after process is killed(Android/iOS)
Kivy Bootstrap for android/ios
Python start, run()
build()
on_start()
on_resume()
Apps functions
External app/os or internal function pauses app
on_pause() Save your work here. Resume is not guaranteed
on_stop()
on_pause()
return False
return True
Kivy Window Destroyed
Resume?
No
Yes
Python stop

2.2.3 Creating the application

The basic notion behind the application is to develop an interface with some animal pictures and when clicked, its takes us to another screen where the corresponding information will be displayed.

-To begin programming with kivy I needed sound knowledge in python program. In kivy programming methodology is same as other object oriented models for app development. but few aspects are different and make it more logical way of appifying your idea.

-To create this application these were the first thoughts

- What data does my application process?
- How do I visually represent that data?
- How does the user interact with that data?

-Initial Structure: main.py

This is a python file which binds everything together.

```
#main.py

from kivy.app import App # importing base app class to inherit app's base
                              class
from kivy.uix.widget import Widget #importing widget class



class MenuScreen(Widget): # defining widget class
    pass



class Animalia(App): #defining base class for kivy application
    def build(self): #function where we initialize & return root widget
        return sm()  #return screen manager object wich we will define later
                      and sm will be root widget of this app



if __name__ == '__main__':
    Animalia().run()  #here where Animalia is initialized and its run()
                              method is called
```

-KV code

The KV language (sometimes called kvlang, or kivy language), allows you to create your widget tree in a declarative way and to bind widget properties to each other or to call-backs in a natural manner. It allows for very fast prototyping and agile changes to your UI. It also facilitates a good separation between the logic of your application and its User Interface.

There are two ways to load Kv code into your application:

- By name convention:

  Kivy looks for a Kv file with the same name as your App class in lowercase, minus "App" if it ends with 'App'. E.g:

  ```
  Animalia -> Animalia.kv.
  ```

If this file defines a *Root Widget* it will be attached to the App’s *root* attribute and used as the base of the application widget tree.

- `Builder`: You can tell Kivy to directly load a string or a file. If this string or file defines a root widget, it will be returned by the method:
- `Builder.load_file('path/to/file.kv')`

or:

- `Builder.load_string(kv_string)`

```
#Animalia.kv


< MenuScreen>:  #base screen for my app
       GridLayout:
```

By this, we have designed the basic structure of our application. now we are going to populate the screen with widgets.

What all we need ?

- Clickable Images/Button
- Threaded screens to show the data
- Images and text to put information

And all this adding widget and stuffs, we will do in our .kv file.

Let's do it

first we have to add a base layout -

- Anchor layout: Adds elements to the border of the window(top, bottom etc..)
- Box layout: Adds elements in either vertical or horizontal format.
- Float layout: Adds element on basis of position vector throughout window
- Grid layout: Adds elements in matrix form(Columns and Rows).
- Page Layout: create a simple multi-page layout
- Relative layout: layout allows you to set relative coordinates for children

We will use **GridLayout** for MenuScreen in order to create a grid of clickable images which will take us to another screen with information.

Here we add it

```
#Animalia.kv
```

```
<MenuScreen>:  #parent screen of application
    GridLayout:
        spacing:10
        padding:10
        cols:2            #defining number of columns
        canvas.before:              #defining a base for our screen widgets
            Color:
                rgba: 1, 1, 1, 1   # changing background to white
            Rectangle:   #the canvas will be rectangle
                pos: self.pos         #position will be default 0,0
                size: self.size        #size will be default screen size
```

Now let's draw something on canvas we created in last step. i.e. we will add buttons and set their on_press() properties.

```
#Animalia.kv



<MenuScreen>:
    GridLayout:
        spacing:10
        padding:10
        cols:2
        canvas.before:
            Color:
                rgba: 1, 1, 1, 1
            Rectangle:
                pos: self.pos
                size: self.size

        Button:
            on_press: root.manager.current = 'settingsA'  #it will take us to widget defined as settingsA
            background_normal: str(False)   #setting button background to off
            #padding:10,10
            #spacing: 10
            Image:
                source: 'a1.png'        #embedding an image over button making it a image button
                allow_stretch: True    #allowing altering stretch property of image
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False     #allowing altering of image ratio
                size:350,400          #defining size of image




        Button:
            on_press: root.manager.current = 'settingsB'
            background_normal: str(False)
            Image:
                allow_stretch: True
                source: 'a2.png'
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False
                size:350,400
        Button:
            on_press: root.manager.current = 'settingsC'
            background_normal: str(False)
            Image:
                source: 'a3.png'
                allow_stretch: True
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False
                size:350,400

        Button:
            on_press: root.manager.current = 'settingsD'
            background_normal: str(False)
            Image:
                allow_stretch: True
                source: 'a4.png'
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False
                size:350,400
        Button:
            on_press: root.manager.current = 'settingsE'
            background_normal: str(False)
            Image:
                source: 'a5.png'
                allow_stretch: True
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False
                size:350,400

        Button:
            on_press: root.manager.current = 'settingsF'
            background_normal: str(False)
            Image:
                source: 'a6.png'
                allow_stretch: True
                y: self.parent.y + self.parent.height - 400
                x: self.parent.x
                size_hint_y: None
                keep_ratio:False
                size:350,400
```

By this, our basic kv work is over but still app doesn't have any proper logic. so for that we have edit our "main.py file".

In that we have to define screen classes so that we can implement screen transitions on button press.

```
#main.py

Builder.load_file('./animalia.kv')   #loading the kv file we created in previous steps

# 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.

# Declare both screens
class MenuScreen(Screen):                       #creating a new screen class for new screen
    pass

class SettingsScreenA(Screen):
    pass
```

```
class SettingsScreenB(Screen):
    pass
class SettingsScreenC(Screen):
    pass
class SettingsScreenD(Screen):
    pass
class SettingsScreenE(Screen):
    pass
class SettingsScreenF(Screen):
    pass
# Create the screen manager
sm = ScreenManager()                                    #defining an object to call ScreenManager
sm.add_widget(MenuScreen(name='menu'))                  #initializing and adding screen widget
sm.add_widget(SettingsScreenA(name='settingsA'))
sm.add_widget(SettingsScreenB(name='settingsB'))
sm.add_widget(SettingsScreenC(name='settingsC'))
sm.add_widget(SettingsScreenD(name='settingsD'))
sm.add_widget(SettingsScreenE(name='settingsE'))
sm.add_widget(SettingsScreenF(name='settingsF'))


class TestApp(App):

    icon = 'custom-kivy-icon.png'
    def build(self):
        return sm


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

now it's time to decorate our different screen in .kv

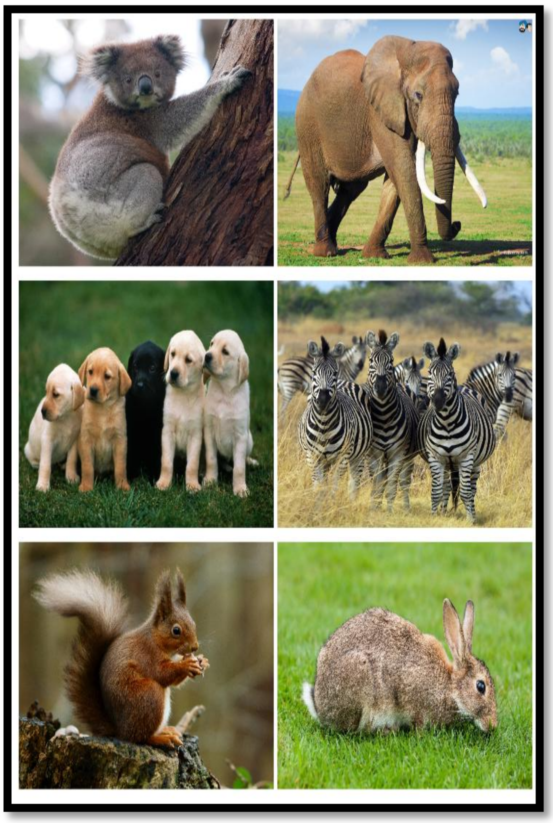
```
#Animalia.kv
<SettingsScreenA>:                                           #decorating Screen A
    BoxLayout:
        spacing:10
        padding:10
        orientation:'vertical'
        canvas.before:
            Color:
                rgba: 1, 1, 1, 1
            Rectangle:
                pos: self.pos
                size: self.size

        Scatter:                                            #using scatter function to scale image


            size: 50,80
            do_rotation: True
            do_scale: True
            do_translation: True
            auto_bring_to_front: False

            Image:                                          #defining new image over button
                source: 'a1.png'
                pos: 100,10
                allow_strech: True
                keep_ratio: False
                size: 500, 500
        ScrollView:                                         #adding scroll mechanism to label text
            Label:                                          #defining new label
                size_hint_y: None
                text_size: self.width, None
                size_hint_y: None
                size: self.texture_size
                height: self.texture_size[1]
                color:0,0,0,0.7
                markup: True                                #allowing use of markup tags
                text: <--content-->"
        Button:
            size_hint_y: None
            size_hint_x: None
            size:650,120
            center_x: root.center_x
            on_press:root.manager.current='menu'
            text:'Go Back!!'
```

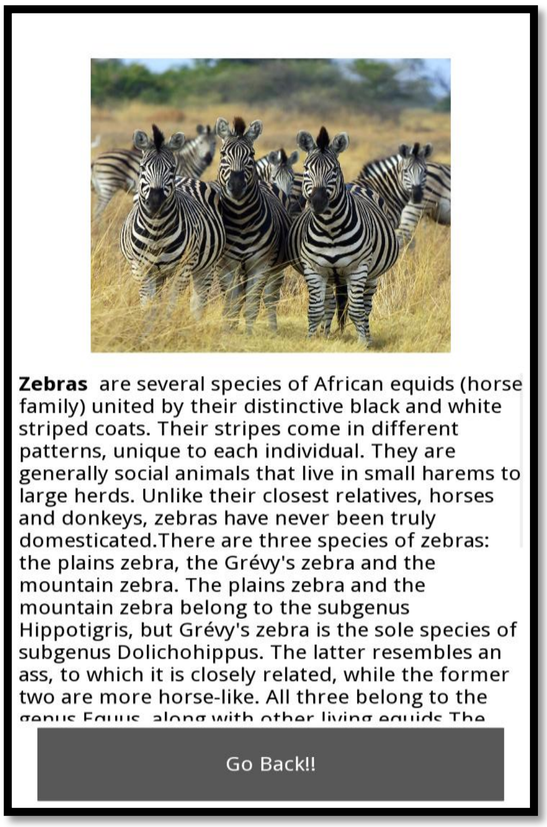
Similarly we will do for all the screens.

In above code to bind button with action we use **on_press()** method

```
Button:
      on_press: root.manager.current = 'settingsC'   #here it is setting the current view as
ScreenC
```

-MenuScreen -ScreenA





Zebras are several species of African equids (horse family) united by their distinctive black and white striped coats. Their stripes come in different patterns, unique to each individual. They are generally social animals that live in small harems to large herds. Unlike their closest relatives, horses and donkeys, zebras have never been truly domesticated.There are three species of zebras: the plains zebra, the Grévy's zebra and the mountain zebra. The plains zebra and the mountain zebra belong to the subgenus Hippotigris, but Grévy's zebra is the sole species of subgenus Dolichohippus. The latter resembles an ass, to which it is closely related, while the former two are more horse-like. All three belong to the genus Equus, along with other living equids. The

2.2.4 Packaging the app for Android

We can package our application with-

- Buildozer
- Kivy App Launcher
- Python-for-Android

We can create a package for android using the Buildozer project. This page explains how to download and use it directly on your own machine, use the prebuilt Kivy Android VM image.

Buildozer

Buildozer is a tool that automates the entire build process. It downloads and sets up all the prequisites for python-for-android, including the android SDK and NDK, then builds an apk that can be automatically pushed to the device.

Buildozer currently works only in Linux, and is an alpha release, but it already works well and can significantly simplify the apk build.
You can get buildozer at https://github.com/kivy/buildozer:

```
>>git clone https://github.com/kivy/buildozer.git
>>cd buildozer
>>sudo python2.7 setup.py install
```

This will install buildozer in your system. Afterwards, navigate to your project directory and run:

buildozer init

This creates a *buildozer.spec* file controlling your build configuration. You should edit it appropriately with your app name etc. You can set variables to control most or all of the parameters passed to python-for-android.
Finally, plug in your android device and run:

buildozer android debug deploy run

to build, push and automatically run the apk on your device.
Buildozer has many available options and tools to help you, the steps above are just the simplest way to build and run your APK. The full documentation is available here. You can also check the Buildozer README at https://github.com/kivy/buildozer.

3. What did I learn ?

3.1. **Technically**

Got familiar with python scripting and its implementation. had hands on experience on kivy development and its libraries.

Appendixes

project codes are available on https://github.com/sterilistic