Lesson 11

Admin Matters

This is Week 10 – Programming quiz is in Week 12

Mini Project – Third Session. Please Bring RPi breadboard and components

Please collect your equipment for 2D week.

Recall OOP

- Instantiation
- __init__ method
- Instance variables / attributes
- Instance methods
- Inheritance

Some Extra (Useful) Python Stuff

A function can have a flexible number of arguments

```
#kwargs is the python convention for "keyworded arguments"
def testing(**kwargs):
    for key in kwargs:
        print key + ' = ' + str(kwargs[key])
testing(a1 = 1, b2 = 4, c3 = 'star')
```

What data type is **kwargs**?

(a) List
(b) Dictionary

Further Reading

Python Script File Special Variable

Every python script file has a special variable called **__name__**

If it is run by itself, it will have a value of **'__main__'**

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

GUI Using Kivy

Terminology

Widget -

Callback -

Binding -

Getting Started

Import the **App** class – the base class for creating kivy Apps.
You use the `run()` method inherited from this class.

Import the class of the widgets that you want to use,
in this case, we want the **`Label`** class.

1. the **`build`** method is
   a. where you initialize your widget's properties.
   b. Bind any callback to your widget (use the **`bind`** method)
   c. Initialize any other variables

Some **`Label`** properties you can use are found in the documentation:
`font_size`
`text`
`color`

2. Define your callback method. Callbacks for mouse clicks would have three inputs:
   a. `self`
   b. `instance` – instance of the widget
   c. `touch` – touch event being recorded

Some of the possible touch variables you can use are

Check the motion of the mouse	Print the position of the mouse
`is_double_tap` **`is_mouse_scrolling`** **`is_touch`** **`dx`** **`dy`**	**`px`** **`py`** **`pos`**

Single-Screen Programming

In the **`build`** method, we do the following

Step 1. Instantiate the root widget - it is usually one of the following **Layout Classes**. Three are given …

`BoxLayout`	https://kivy.org/docs/api-kivy.uix.boxlayout.html
`GridLayout`	https://kivy.org/docs/api-kivy.uix.gridlayout.html
`FloatLayout`	https://kivy.org/docs/api-kivy.uix.floatlayout.html

Step 2. Instantiate other widgets that you want. Basic widgets you should know

`Label`	https://kivy.org/docs/api-kivy.uix.label.html
`Button`	https://kivy.org/docs/api-kivy.uix.button.html
`TextInput`	https://kivy.org/docs/api-kivy.uix.textinput.html

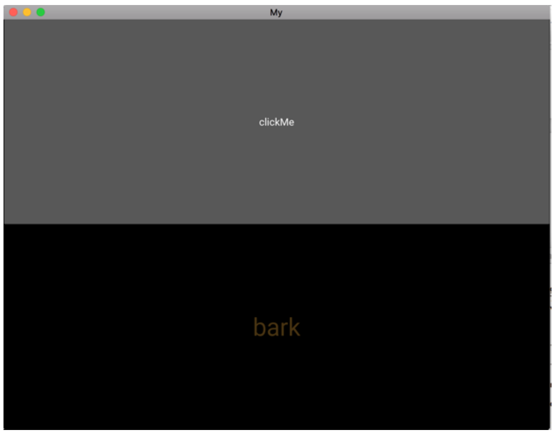
Step 3. Bind any widgets to the callback that you want. Callbacks for button clicks would have two inputs:

a. **`self`**
b. **`instance`** – instance of the widget

Step 4. Add these widgets to the layout in the order that you want. At this stage you should run your python code so as to check for any problems with your layout.

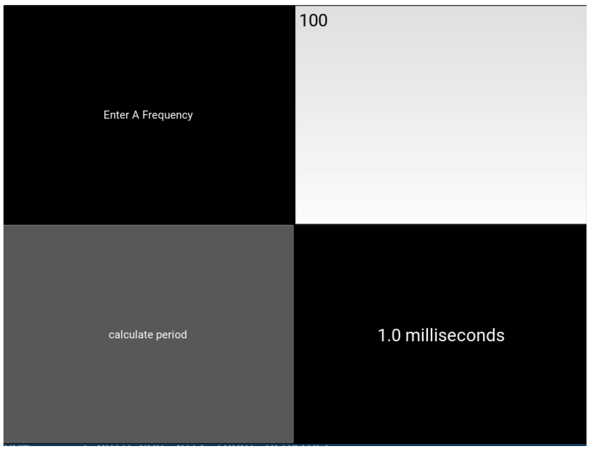
Step 5. Write your callback.

Example 1 - Click a button to display a random word below



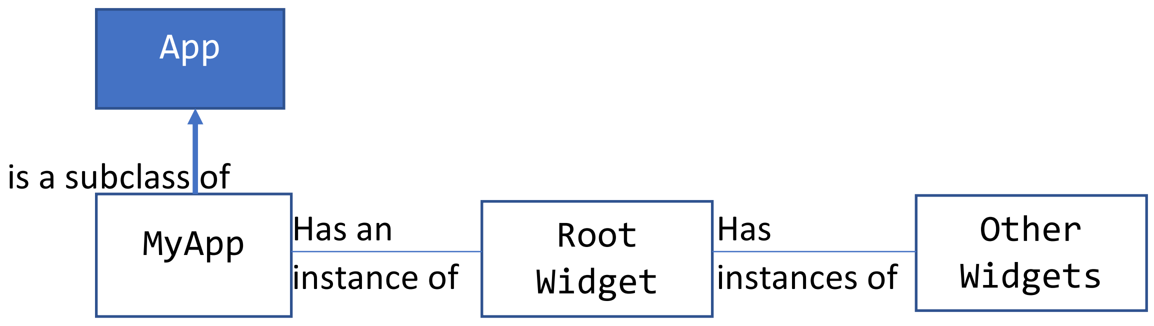
```
from kivy.app import App

#TO DO 0.9  Import necessary widget classes

import random


class MyApp(App):


    ## TO DO 1.0 Create your layout in this method

    def build(self):

        ## TO DO 1.1 Create an instance of the BoxLayout Class

        ## TO DO 1.2 Create an instance of a Button with font size

        ## TO DO 1.3 Create an instance of a Label. This must be a instance
variable so that it can be accessed in the callback

        ## TO DO 1.4 Bind the Label to the callback function

        ## TO DO 1.5 Add the widgets in sequence to the root widget

        ## TO DO 1.6 Return the root widget


    ## TO DO 2.0 Write your callback

    def change(self,instance):

        ## TO DO 2.1 Make a list of the words

        ## TO DO 2.2 Write code to randomly select an element from the list and
assign it to the text attribute of the label

        ## TO DO 2.3 Write code to randomly generate a color and assign it to the
color attribute of the

if __name__ == '__main__':

    MyApp().run()
```

Example 2 - Simple Calculation App



```
from kivy.app import App

#TO DO 0.9 Import statements


class MyApp(App):

    ## TO DO 1.0 Create your layout in this method

    def build(self):

        ## TO DO 1.1 Create an instance of the GridLayout with two columns

        ## TO DO 1.2 Create an instance of a Button

        ## TO DO 1.3 Create instances of Label. The Label object that displays the
result must be an instance variable.

        ## TO DO 1.4 Bind the Button to the callback function

        ## TO DO 1.5 Create an instance of the TextInput

        ## TO DO 1.6 Add the widgets in sequence to the root widget

        ## TO DO 1.7 Return the root widget


    ## TO DO 2.0 Write your callback

    def change(self,instance):

        ##TO DO 2.1 Access the text attribute of your TextInput

        ##TO DO 2.2 Convert it to float object

        ##TO DO 2.2a (optional) perform the necessary checks

        ##TO DO 2.3 Make the calculation

        ##TO DO 2.4 Assign the output to the text attribute of the appropriate
label

if __name__ == '__main__':

    MyApp().run()
```

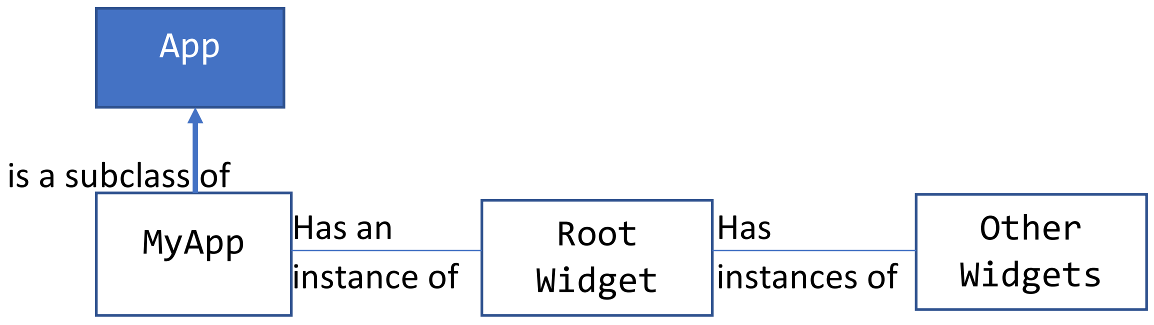
Class Relationships - Single Screen



Your Root Widget is an instance of a Layout (e.g. `BoxLayout`)

And `BoxLayout` would contain instances of other Widgets

Multiple Screens



In the case of multiple screens, your Root Widget is an instance of `ScreenManager`