Introduction to kivy

Ashley DaSilva

Today's materials:
Github (https://github.com/adasilva/intro_kivy)
Or Flash drive (floating around the room)

Try running a test kivy app!
Linux: python test.py
Mac: kivy test.py
Windows: right click on test.py, choose send to kivy

What is kivy?

- Framework for cross-platform apps
- Logic for how the app looks
- Building blocks for designing the app

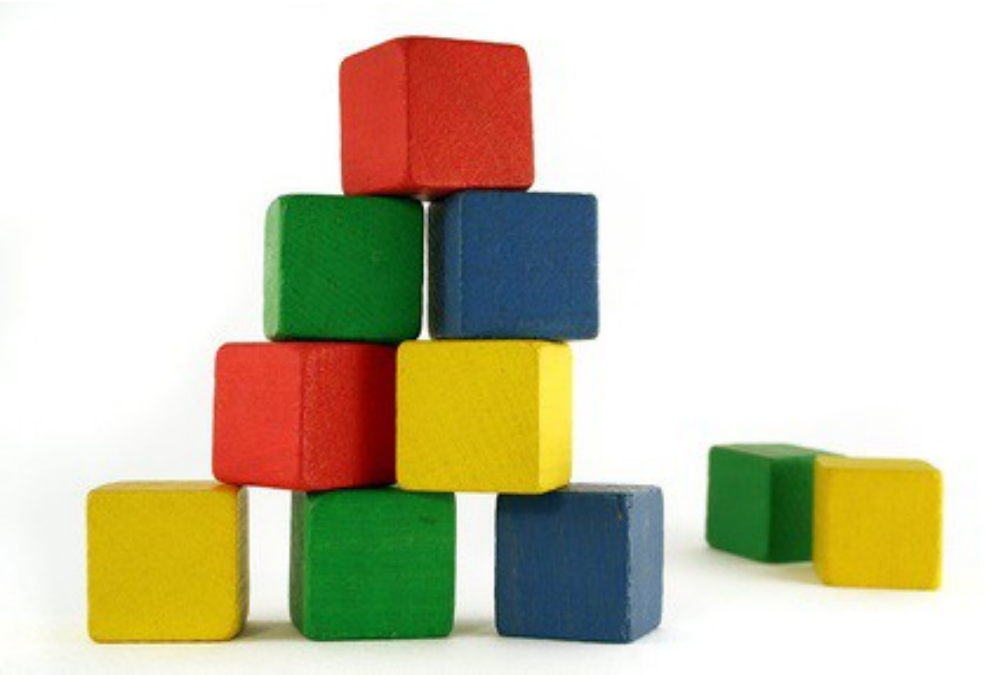


Image source: http://www.getfilecloud.com/blog/2014/01/the-fundamental-building-blocks-of-cloud-computing/#.VB3BtnX7HeQ

Theme for today: workout tracker

Ideas:

- Plan your workouts for the week
- Check them off when complete
- Include the amount of time spent, day of the week, etc
- Badges when you achieve goals
- Analyze your personal workout history

The building blocks of kivy apps

- The App class
- Widgets
  - Layouts
  - **User interface widgets**

Images from the kivy documentation: http://kivy.org/docs/

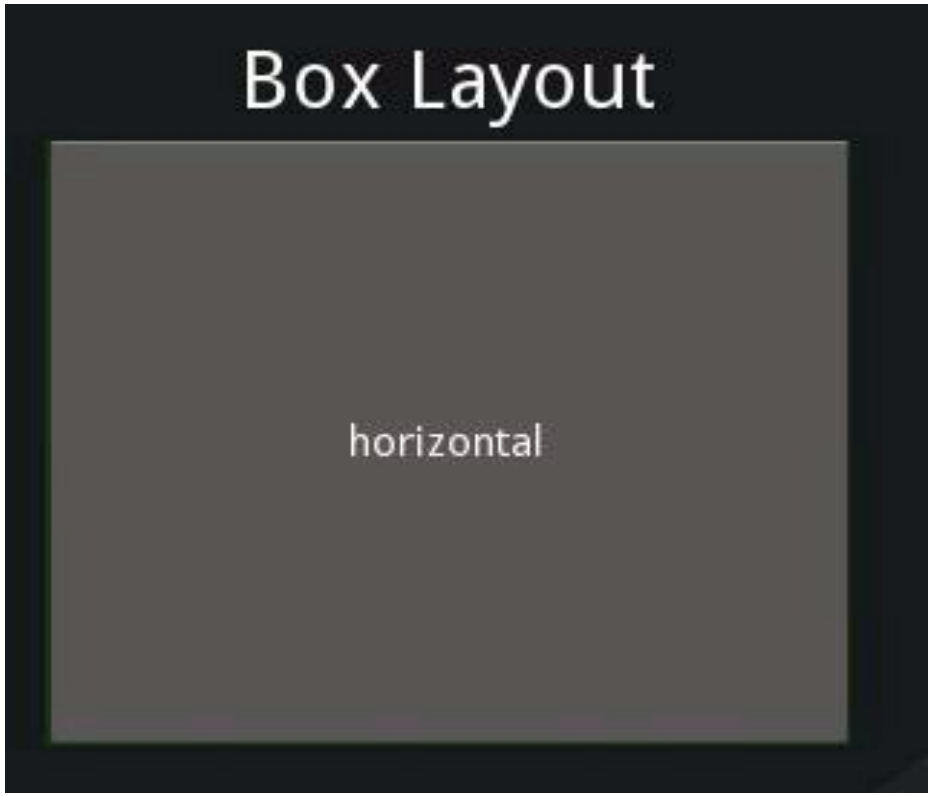
Layouts

- Provide overall organization
- Can be nested
- Available layouts:
  - Grid
  - Box
  - Float
  - See more in kivy documentation

http://kivy.org/docs/gettingstarted/layouts.html

Layouts

- Provide overall organization
- Can be nested
- Available layouts:
  - Grid
  - Box
  - Float
  - See more in kivy documentation



http://kivy.org/docs/gettingstarted/layouts.html

Let's start building our app!

- Enter 01_buildingblocks folder

Let's start building our app!

- Enter 01_buildingblocks folder
- Import from kivy

```
from kivy.app import App

from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
```

Let's start building our app!

- Enter 01_buildingblocks folder
- Import from kivy

```
from kivy.app import App

from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
```

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        # add widgets...
```

Let's start building our app!

- Enter 01_buildingblocks folder
- Import from kivy

```
from kivy.app import App

from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
```

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        # add widgets...
```

```
class WorkoutApp(App):
    def build(self):
        return MainScreen()
```

Adding widgets

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
        self.cols=3

        l1=Label(text='Workout')
        l2=Label(text='Day')
        l3=Label(text='Time')

        in1=TextInput(multiline=False
        in2=TextInput(multiline=False
        in3=TextInput(multiline=False)

        self.add_widget(l1)
        self.add_widget(l2)
        self.add_widget(l3)
        self.add_widget(in1)
        self.add_widget(in2)
        self.add_widget(in3)
```

Implements features of the base class (grid layout)

Adding widgets

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
        self.cols=3

        l1=Label(text='Workout')
        l2=Label(text='Day')
        l3=Label(text='Time')

        in1=TextInput(multiline=False
        in2=TextInput(multiline=False
        in3=TextInput(multiline=False)

        self.add_widget(l1)
        self.add_widget(l2)
        self.add_widget(l3)
        self.add_widget(in1)
        self.add_widget(in2)
        self.add_widget(in3)
```

Choose number of columns for the grid layout

Adding widgets

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
        self.cols=3

        l1=Label(text='Workout')
        l2=Label(text='Day')
        l3=Label(text='Time')

        in1=TextInput(multiline=False
        in2=TextInput(multiline=False
        in3=TextInput(multiline=False)

        self.add_widget(l1)
        self.add_widget(l2)
        self.add_widget(l3)
        self.add_widget(in1)
        self.add_widget(in2)
        self.add_widget(in3)
```

Set up three label widgets – but to add them to the layout...

Adding widgets

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
        self.cols=3

        l1=Label(text='Workout')
        l2=Label(text='Day')
        l3=Label(text='Time')

        in1=TextInput(multiline=False
        in2=TextInput(multiline=False
        in3=TextInput(multiline=False)

        self.add_widget(l1)
        self.add_widget(l2)
        self.add_widget(l3)
        self.add_widget(in1)
        self.add_widget(in2)
        self.add_widget(in3)
```

...Need to use the add_widget method (from Widget class)

Running the app

- Kivy App has a run() method

```
class WorkoutApp(App):
    def build(self):
        return MainScreen()
```

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

Task: add more widgets

- Try adding more text input boxes or labels

Image originally from: http://hyperboleandahalf.blogspot.com/2010/06/this-is-why-ill-never-be-adult.html

Separate design from main code using kivy language

- Setting up and adding widgets is repetitive!
- Can be hard to read for more complicated apps

- Enter 02_kivylanguage folder
- Open workout.kv and workoutApp.py

Basics of kivy language

- Kivy looks for a .kv file with the same name as your app, minus the App:
  - class WorkoutApp → workout.kv
- Each custom widget class is defined in the .kv
  - Keep the class definition in main.py!
  - Don't need to import widgets defined in .kv
  - MainScreen(GridLayout) → <MainScreen>
- Customize kivy widgets throughout the entire app
  - e.g. set multiline=False for all text inputs

The .kv file

- Widgets get defined and added in one step!

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
```

```
<MainScreen>:
    cols: 3
    Label:
        text: 'Workout'
        size_hint_x: 2/3.
    Label:
        text: 'Day'
        size_hint_x: 1/6.
```

The entire MainScreen definition in main.py!

The .kv file

- Widgets get defined and added in one step!

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
```

```
<MainScreen>:
    cols: 3
    Label:
        text: 'Workout'
        size_hint_x: 2/3.
    Label:
        text: 'Day'
        size_hint_x: 1/6.
```

Part of the MainScreen definition in workout.kv

The .kv file

- Widgets get defined and added in one step!

```
class MainScreen(GridLayout):
    def __init__(self,**kwargs):
        super(MainScreen,self).__init__(**kwargs)
```

```
<MainScreen>:
    cols: 3
    Label:
        text: 'Workout'
        size_hint_x: 2/3.
    Label:
        text: 'Buy'
        size_hint_x: 1/6.
```

Size hints: tell horizontal (x) and vertical (y) size compared to parent

The .kv file

- Global definitions are easy to set

```
TextInput:
    multiline: False
```

Sets multiline to False for all TextInput

Task: add your widgets to .kv file!

- Use the .kv file to add more widgets
- Add a 4th column with checkboxes (CheckBox)
- Try out different size hints

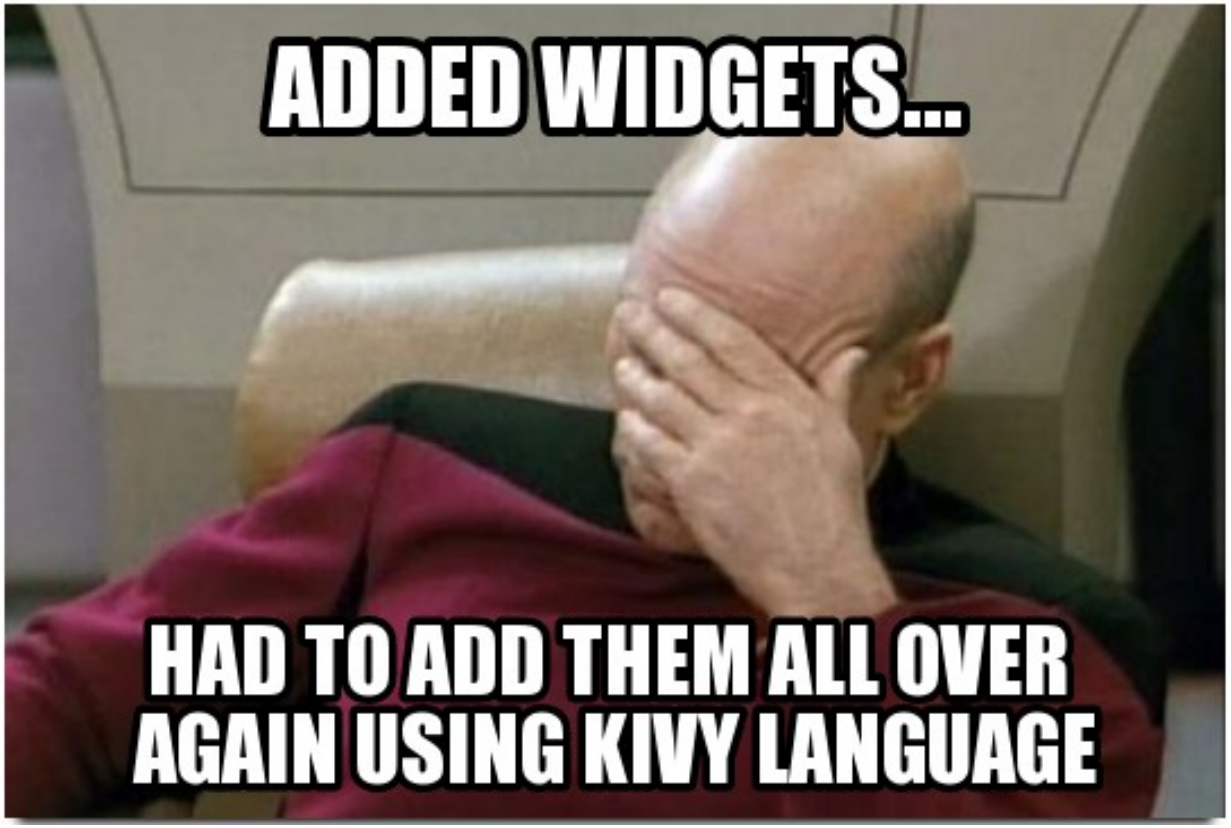


Image originally from: Star Trek the Next Generation, episode “Deja Q”

Button widget and function binding

- Let's make a button!
- When pressed, data from user input is saved to a text file
- This is called **binding**
- Open 03_buttonsAndBinding

Save button

In workout.kv

```
Button:
    text: 'Save'
    on_press: root.save()
```

root refers back to the python class, which must have the save method!

In workoutApp.py

```
def save(self):
    '''Saves the data from the input to a text file.
It is bound to the save button'''
    # do stuff here!
```

Kivy ids

In workout.kv

```
TextInput:
    id: workoutInput1
    size_hint_x: 5/8.
```

Use self.ids to refer back to <MainScreen> from the python code

In workoutApp.py, save function

```
status1 = self.ids.checkbox1.active
workout1 = self.ids.workoutInput1.text
day1 = self.ids.dayInput1.text
time1 = self.ids.timeInput1.text
```

Kivy ids

In workout.kv

```
CheckBox:
    id: checkbox1
    size_hint_x: 1/8.

TextInput:
    id: workoutInput1
    size_hint_x: 5/8.
```

Any property of the widget can be accessed!

In workoutApp.py, save function

```
status1 = self.ids.checkbox1.active
workout1 = self.ids.workoutInput1.text
day1 = self.ids.dayInput1.text
time1 = self.ids.timeInput1.text
```

Kivy ids

In workout.kv

```
CheckBox:
    id: checkbox1
    size_hint_x: 1/8.

TextInput:
    id: workoutInput1
    size_hint_x: 5/8.
```

Any property of the widget can be accessed!

In workoutApp.py, save function

```
status1 = self.ids.checkbox1.active
workout1 = self.ids.workoutInput1.text
day1 = self.ids.dayInput1.text
time1 = self.ids.timeInput1.text
```

Writing the data

```
    def save(self):
        '''Saves the data from the input to a text file.
    It is bound to the save button'''
        workout = self.ids.workoutInput.text
        day = self.ids.dayInput.text
        time = self.ids.timeInput.text
        with open('workoutData.txt','a') as f:
            f.write('%s, %s, %s\n' %(workout, day, time))

        return None
```

- With/as statement
  - Equivalent to f = open('workoutData.txt','a')
  - But makes sure the file gets closed, even if there's an error!

Task: add up total time!

- Use a new button to add up the total time for completed workouts, and display below the time column

Image credit:

Laney Griner

Remove repetitive code with new customized class

- Each workout has a similar format
  - Workout, day, time
  - Clue that maybe it can be simplified!
- Open 04_nestedLayouts folder

WorkoutLayout class

In workout.kv

```
<WorkoutLayout>:
    size_hint_x: 1
    orientation: 'horizontal'
    CheckBox:
        id: status
        size_hint_x: 1/8.
    TextInput:
        id: workoutInput
        size_hint_x: 5/8.
    TextInput:
        id: dayInput
        size_hint_x: 1/8.
    TextInput:
        id: timeInput
        size_hint_x: 1/8.
```

Custom WorkoutLayout includes all text input & can be reused!

In workoutApp.py

```
from kivy.uix.boxlayout import BoxLayout

class WorkoutLayout(BoxLayout):
    pass
```

WorkoutLayout class

In workout.kv

```
<WorkoutLayout>:
    size_hint_x: 1
    orientation: 'horizontal'
    CheckBox:
        id: status
        size_hint_x: 1/8.
    TextInput:
        id: workoutInput
        size_hint_x: 5/8.
    TextInput:
        id: dayInput
        size_hint_x: 1/8.
    TextInput:
        id: timeInput
        size_hint_x: 1/8.
```

Make sure to import BoxLayout at the top of python file

In workoutApp.py

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

```
class WorkoutLayout(BoxLayout):
    pass
```

Using the new layout

In workout.kv

```
<MainScreen>:
    orientation: 'vertical'
    BoxLayout:
        orientation: 'horizontal'
        Label:
            text: 'Workout'
            size_hint_x: 2/3.
        Label:
            text: 'Day'
            size_hint_x: 1/6.
        Label:
            text: 'Time'
            size_hint_x: 1/6.
```

```
    WorkoutLayout:
        id: workout1
    WorkoutLayout:
        id: workout2
    WorkoutLayout:
        id: workout3
    WorkoutLayout:
        id: workout4
    WorkoutLayout:
        id: workout5
    WorkoutLayout:
        id: workout6
    WorkoutLayout:
        id: workout7
```

In workoutApp.py, changed MainScreen to box layout!

```
class MainScreen(BoxLayout):
```

Nested layouts, nested ids

- The nested layouts means there are also nested ids:

```
def save(self):
    '''Saves the data from the input to a text file.
It is bound to the save button'''
    workout = self.ids.workout1.ids.workoutInput.text
    day = self.ids.workout1.ids.dayInput.text
    time = self.ids.workout1.ids.timeInput.text
    with open('workoutData.txt','a') as f:
        f.write('%s, %s, %s\n' %(workout, day, time))

    return None
```

Refers to the id in the MainScreen class (because of the self)

Nested layouts, nested ids

- The nested layouts means there are also nested ids:

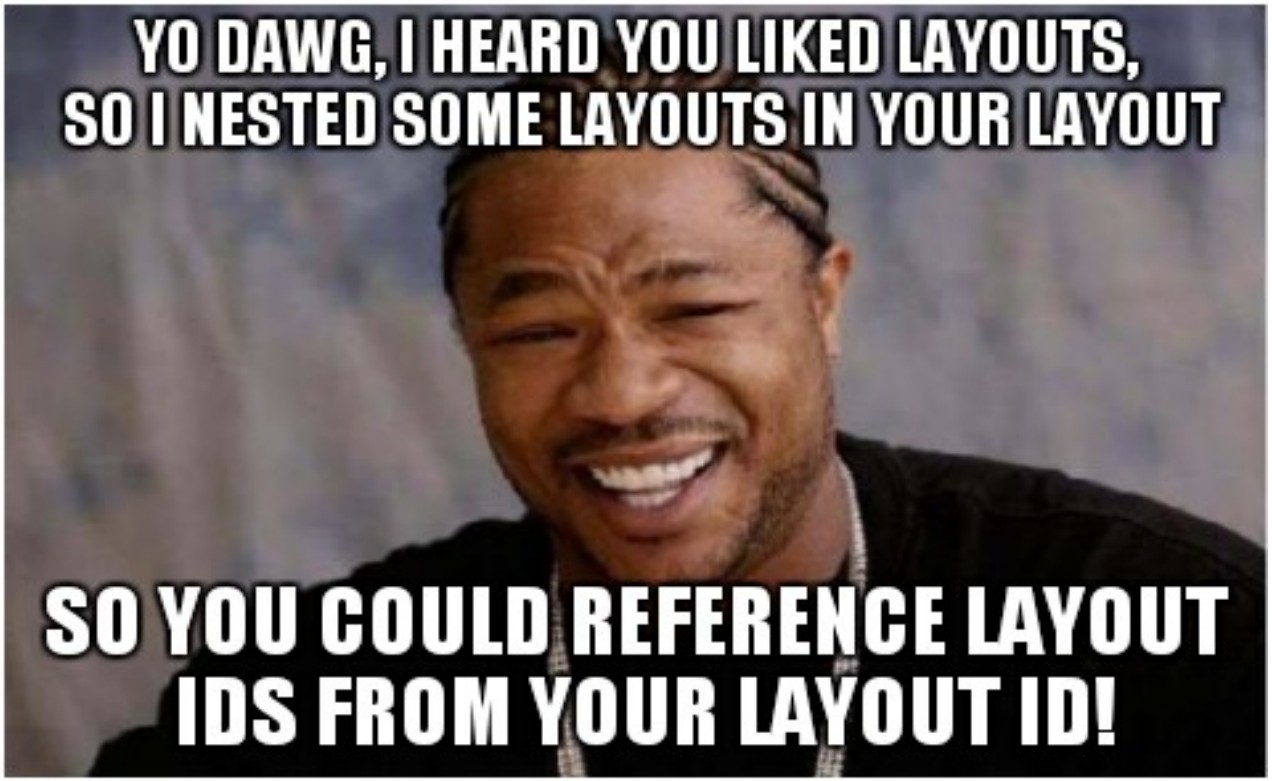
```
<WorkoutLayout>:
    size_hint_x: 1
    orientation: 'horizontal'
    CheckBox:
        id: status
        size_hint_x: 1/8.
    TextInput:
        id: workoutInput
        size_hint_x: 5/8.
    TextInput:
        id: dayInput
        size_hint_x: 1/8.
    TextInput:
        id: timeInput
        size_hint_x: 1/8.
```

```
the input to a text file.
tton'''
kout1 ids.workoutInput.text
..ids.dayInput.text
t1.ids.timeInput.text
txt','a') as f:
s\n' %(workout, day, time))
```

Each instance of WorkoutLayout has its own set of ids!

Tasks:

- Right now the save function only saves the first workout. Revise it to loop over all workouts.
- Add the total button for the workout time. Hint: use nested layout in bottom row.



http://knowyourmeme.com/memes/xzibit-yo-dawg

Take it a step further:

- Add a dropdown menu for the day of the week
  - Dropdown menu is really just a container for buttons!
  - Must be attached to a button
  - When button is pressed, dropdown menu appears
- Use a database instead of a text file to save data
- Your own ideas?

Packaging for android

- Only in linux (for now)
  - Offered by kivy: Kivy android virtual machine
- Python for android project
- Buildozer
  - Main python file must be called main.py
  - The first time, it will download extra packages (android SDK and android NDK)
  - Deals with python dependencies for you!
  - Produces a .apk which can be installed directly on android

Packaging for android

- Run: `buildozer init`
- A new file is created: `buildozer.spec`
  - Contains all the info needed to build the android package
  - Open the example in the main directory to check it out!
- To build the package, run: `buildozer android debug deploy run`
  - Expect it to take some time
  - Grab android packages in the bin folder, version number matches section number!