learn.Kivy()

A Python user's guide to using Kivy GUI Framework

Kevin Xie

Hello!

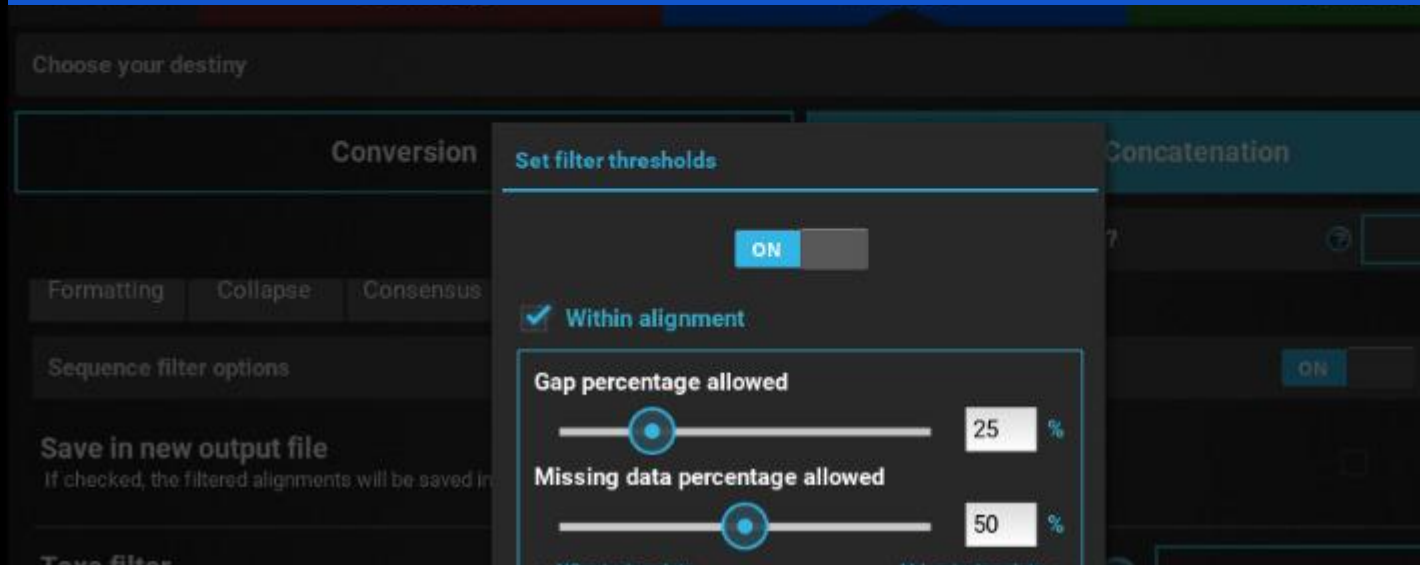
I am Kivy

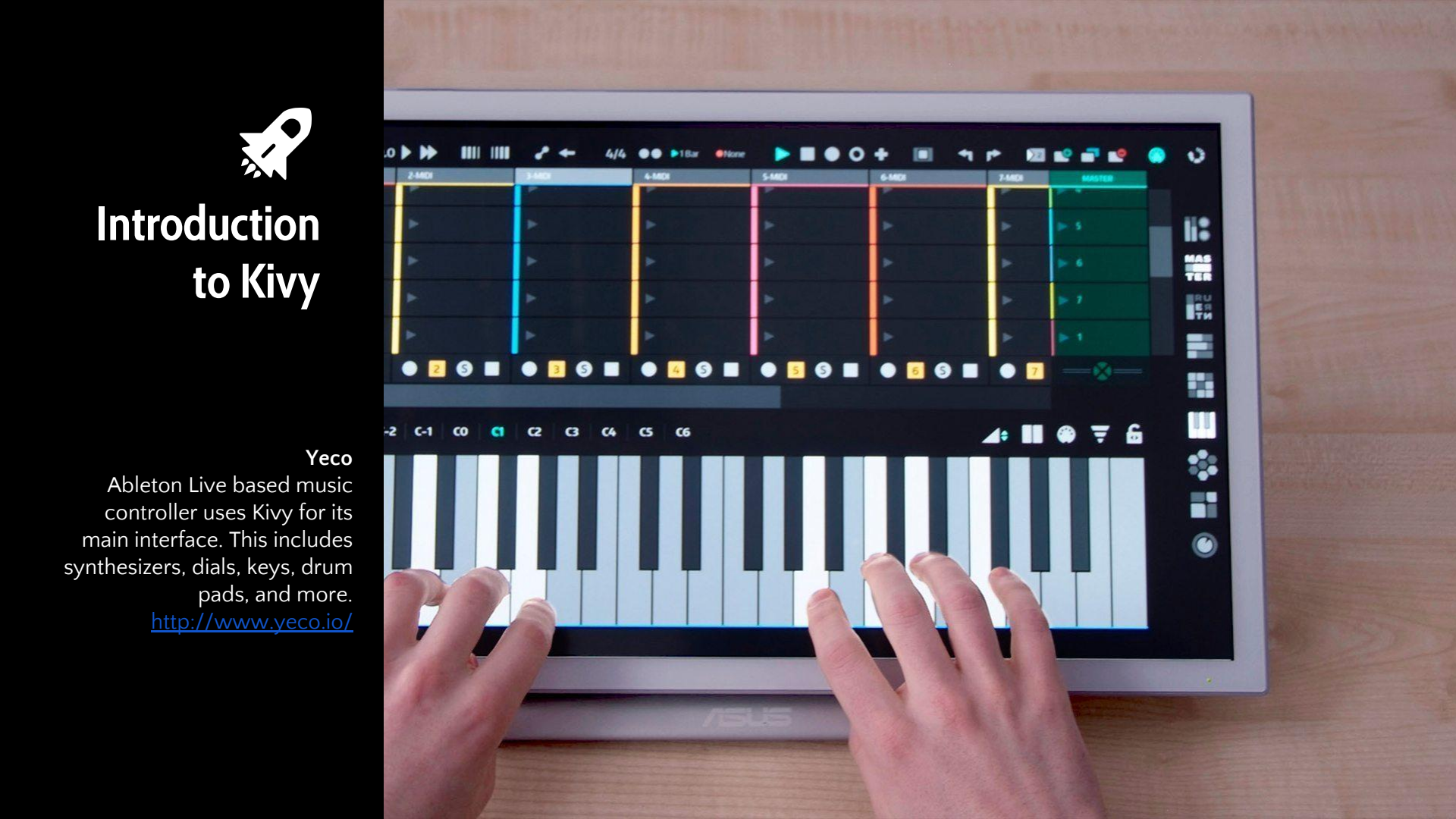
I’m an NUI Framework that can help you create a simple, cross-platform Natural User Interface for your program.

Introduction to Kivy

Kivy is a handy cross-platform NUI framework for any aspiring Python programmer. It can help get a simple mobile, desktop, or remote application off the ground quickly. In order to fully utilize it, there are a few things to master, including a whole new language! Learning the syntax of Kv can be essential in using the framework to its maximum potential. However, it's possible to write a Kivy-based application without ever typing a word in Kv. This book will introduce you to Kivy and its position in your workflow, teach you the basic syntax and advantages of using Kv over Python when programming your NUI, offer design advice for the optimal user experience, and provide a good reference resource for future use.

TriFusion
Phylogenomic data gathering, processing and visualization software uses Kivy to handle the values and options for manipulating multiple sequence alignments.
https://odiogosilva.github.io/TriFusion/





Introduction to Kivy

Yeco

Ableton Live based music controller uses Kivy for its main interface. This includes synthesizers, dials, keys, drum pads, and more.

http://www.yeco.io/

Installing Kivy

INSTALLING ON macOS

1. Download the kivy.app from:

   https://kivy.org/#download

2. Copy the file to Applications.
3. Run `makesymlinks` in the window that opens when you open the dmg.

INSTALLING ON Windows

Open Command Prompt by typing cmd into the Start Menu and run the following commands in order:

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

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

python -m pip install kivy.deps.gstreamer

python -m pip install kivy
```

For a more in-depth tutorial on how to install Kivy, visit https://kivy.org/doc/stable/installation/installation.html

More modules and dependencies are available to install through pip on the website.

Installing Kivy

```
<Catalog>:
    language_box: language_box
    screen_manager: screen_manager
    auto_reload: chkbx.active
    info_label: info_lbl
    orientation: 'vertical'
    BoxLayout:
        padding: '2sp'
        canvas:
            Color:
                rgba: 1, 1, 1, .6
            Rectangle:
                size: self.size
                pos: self.pos
        size_hint: 1, None
        height: '45sp'
        Spinner:
            size_hint: None, 1
            width: '108sp'
            text: 'Welcome'
            values: [screen.name for screen in screen_manager.screens]
            on_text: root.show_kv(*args)
        Widget:
        BoxLayout:
            size_hint: None, 1
            width: '150sp'
            Label:
                text: "Auto Reload"
            CheckBox:
                id: chkbx
                active: True
                size_hint_x: 1
        Button:
            size_hint: None, 1
            width: '108sp'
            text: 'Render Now'
            on_release: root.change_kv(*args)
    BoxLayout:
        id: reactive_layout
        orientation: 'vertical' if self.width < self.height else 'horizontal'
```

INSTALLING THE KV SYNTAX ON Sublime Text

By default, Sublime does not recognize a .kv file. As such, it is highly recommended to install a custom Kv language definition for increased legibility. The Kivy Language (Kv) syntax can be installed through Package Control:

1. Press CTRL(CMD on macOS) +SHIFT+P
2. Enter “install”
3. Search for “Kivy Language” and install the top result

For a more in-depth tutorial on how to install Kivy, visit https://kivy.org/doc/stable/installation/installation.html

More modules and dependencies are available to install through pip on the website.

1

Document Setup

Ensuring the most efficient workflow

Creating two documents

When setting up your program's environment, it is important to have a separate .kv and .py file. Your .py file will be the home of your program and its logic, and the .kv will be the home of your NUI and its scripted behavior.

Python

As usual, save a file as [name].py in your desired directory. This will serve as your programs muscles.

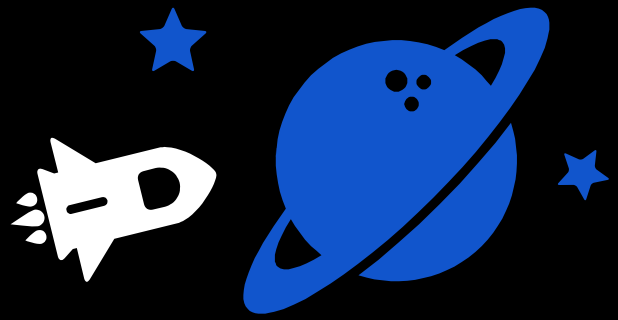
Kv

Save this file in the same directory as your python file in the format [name].kv. This will serve as the "skin" and "skeleton" of your program.

Creating two documents

helloworld.kv

helloworldkv.py



2: The Kv Syntax

How to eat, sleep, and think in Kv.

Why use Kv?

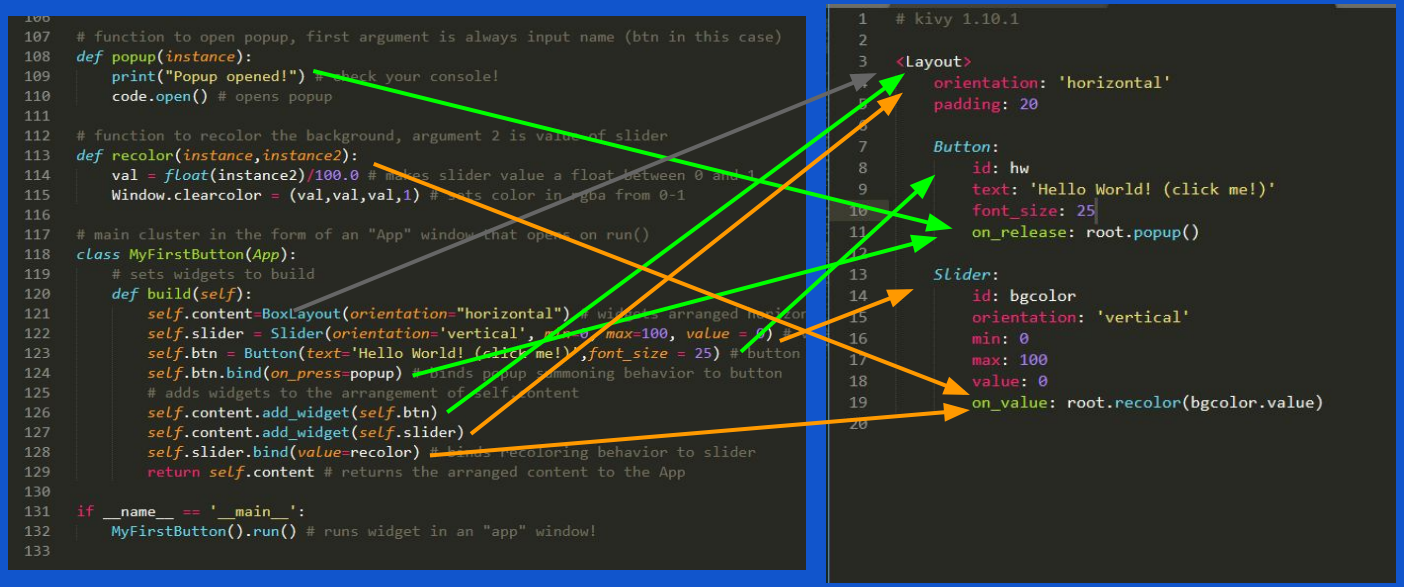
Example files
Don't forget to download the provided example files at the repository below.
http://www.githb.com/suddenlykevin/cs550library

You might be confused as to why you would want to learn a new language over Python for a simple framework. Kivy's native language, Kv, offers a much more streamlined way to define UI elements (called widgets).

This means not only that code for behavior (python) can be separated from code for appearance (Kivy), but that the code for appearance will be much more legible and editable in the debugging stage.

One reason for this is that, as you will see on the next page, Kivy groups a widget's properties directly underneath said widget, whereas Python can complicate this because where certain properties are defined matter to the logic of the code.

Python vs. Kv



```python
# function to open popup, first argument is always input name (btn in this case)
def popup(instance):
    print("Popup opened!") # check your console!
    code.open() # opens popup

# function to recolor the background, argument 2 is value of slider
def recolor(instance,instance2):
    val = float(instance2)/100.0 # makes slider value a float between 0 and 1
    Window.clearcolor = (val,val,val,1) # sets color in rgba from 0-1

# main cluster in the form of an "App" window that opens on run()
class MyFirstButton(App):
    # sets widgets to build
    def build(self):
        self.content=BoxLayout(orientation="horizontal") # widgets arranged horizontally
        self.slider = Slider(orientation='vertical', min=0, max=100, value = 0) # slider
        self.btn = Button(text='Hello World! (click me!)',font_size = 25) # button
        self.btn.bind(on_press=popup) # binds popup summoning behavior to button
        # adds widgets to the arrangement of self.content
        self.content.add_widget(self.btn)
        self.content.add_widget(self.slider)
        self.slider.bind(value=recolor) # binds recoloring behavior to slider
        return self.content # returns the arranged content to the App

if __name__ == '__main__':
    MyFirstButton().run() # runs widget in an "app" window!
```

```
# kivy 1.10.1

<Layout>
    orientation: 'horizontal'
    padding: 20

    Button:
        id: hw
        text: 'Hello World! (click me!)'
        font_size: 25
        on_release: root.popup()

    Slider:
        id: bgcolor
        orientation: 'vertical'
        min: 0
        max: 100
        value: 0
        on_value: root.recolor(bgcolor.value)
```

Above, the Python code for a Kivy UI is converted into Kv. This clearly illustrates how much more efficiently Kv groups, defines, and organizes widgets and their properties. This is the main reason that most developers decide to use Kv over Python for their UI.

Quick Reference: Modules

To get started, in order to use the widgets and other functionality provided by Kivy, it is important to know what modules must be imported:

Always Essential:

```
import kivy # imports kivy's core functionality

from kivy.app import App # imports ability to create "app windows" on mac OS and PC

from kivy.lang import Builder # imports the ability to read and build programs based on the Kivy language
```

Quick Reference: Modules

Using Kivy can mean importing so many submodules that the top of your code looks like a mountain, but it's worth it. Here are a few commonly used modules. (Find a full list at https://kivy.org/doc/stable/api-kivy.html)

from kivy.core.window import Window	For altering characteristics of the app window (e.g., color in rgba)
from kivy.uix.boxlayout import BoxLayout	For organizing widgets in the traditional "box" layout.
from kivy.uix.label import Label	Labels are used for any read-only text in your app -- including titles, descriptions, and instructions.
from kivy.uix.image import Image	For opening images within Kivy.
from kivy.uix.button import Button	For buttons and button behavior in Kivy.
from kivy.uix.slider import Slider	For sliders and slider behavior in Kivy.

Linking your Kv file

It is possible to program with Kivy using entirely Python, but this textbook will use Kv in its examples to help users get into the habit of programming in Kv.

To start, link your Python and Kv files, use the following:

```
from kivy.lang import Builder # this will import the kv language compiler.
Builder.load_file('[name].kv') # this will actually link your kv file.
```

Alternatively, you can link your .kv file in your App class, as such:

```
class FilterApp(App):
    def build(self):
        self.load_kv('my.kv') # links kv file
```

Now you can start to write new classes of widgets in Kv to be used in your Python code.

Quick Reference: Setup

Alright, quit dilly-dallying, let's get started coding! Now that you've linked the Kv to your python file and imported the relevant modules, it's time to setup the code for execution:

```python
import kivy
from kivy.app import App
from kivy.core.window import Window

kivy.require('1.10.1')

from kivy.uix.button import Button
from kivy.uix.image import Image
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.gridlayout import GridLayout
from kivy.uix.slider import Slider
from kivy.uix.filechooser import FileChooser
from kivy.uix.behaviors import ToggleButtonBehavior
from kivy.uix.checkbox import CheckBox
from kivy.uix.accordion import Accordion
from kivy.uix.spinner import Spinner



class ClassName(BoxLayout):

    def behavior(self):
        print(self.ids.slider.value)

    def behaviorTwo(self,value):
        print(value)

class AppWindow(App):
    def build(self):
        self.load_kv('kvName.kv')
        return ClassName()

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

Python File

```
# kivy 1.10.1

<ClassName>
    orientation: 'propertyHere'

    Button:
        id: button
        text: 'Hello!'
        on_release: root.behavior()

    Slider:
        id: slider
        min: 0
        max: 100
        on_value: root.behaviorTwo(self.value)
```

Kv File

Quick Reference: Layouts

If you followed the Windows instructions to install Kivy, you should have a library of example files at a path similar to:

```
C:\Users\[username]\AppData\Local\Programs\Python\Python37\share\kivy-examples\
```

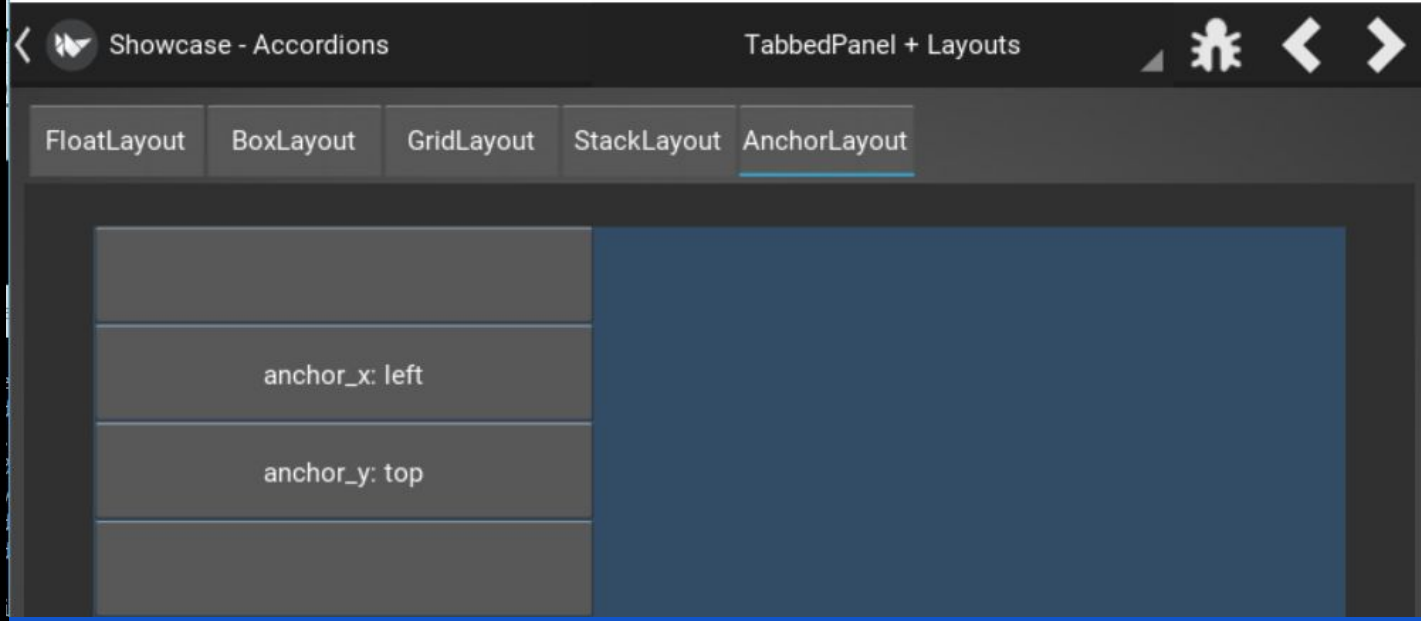
However, if you installed using macOS, please follow the online instructions (https://kivy.org/doc/stable/installation/installation.html) on how to download and access these example files.

These example files are a great reference for any Kivy component, but this textbook will focus on the main.py file in kivy-examples > demo > showcase. Open this file using:

```
py directory://main.py # replace directory with your directory
```

Within the window, there will be a dropdown menu. Select “TabbedPanel + Layouts.” This visual helps to showcase each of the different available layout “modes” in Kivy. These are ways to organize your widgets on your user’s screen.

Quick Reference: Layouts



BoxLayout is the traditional layout, where widgets can be stacked uniformly horizontally or vertically. GridLayout uses a grid with predefined columns or rows to organize widgets. StackLayout resembles BoxLayout but is not uniform. FloatLayout and AnchorLayout are based on coordinates or relative locations.

Which layout style you choose is based on personal preference, and it is important to remember that layouts can be nested within each other to create more complex GUIs.

Quick Reference: Setup

The most important elements to focus on in the Python file:

```
kivy.require('1.10.1') # this will ensure compatibility with other systems.
```

```
class ClassName(BoxLayout) # This calls on the class defined in the Kv code and defines the widget type of the root widget (BoxLayout). Nested within are the different behaviors that are triggered in the Kv widget.
```

```
Class AppWindow(App) # this will open the App Window. Use the builder() function to call on your root widget class. In order to open the window, you must run it using:
```

```
AppWindow.run()
```

These elements are present in almost every Kivy-Python file (with varying names, of course.)

This will also be your first time reading Kv. Very simple, right? It resembles HTML or CSS in that properties are grouped together and well-defined.

Each widget gets its own indentation and nested properties/other widgets.

Later, we will provide a list of common properties and behaviors.

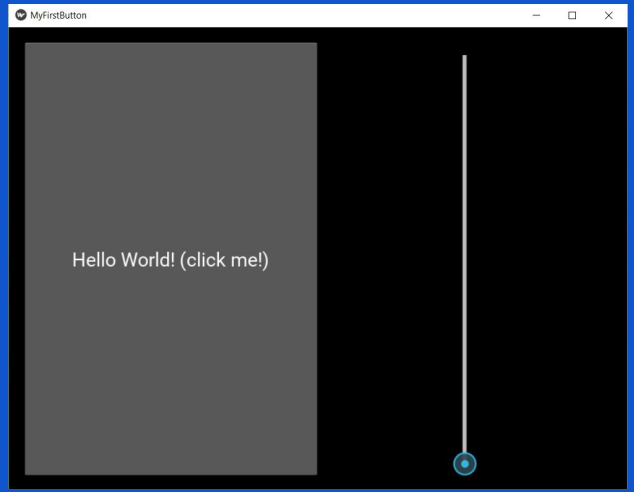
The example files

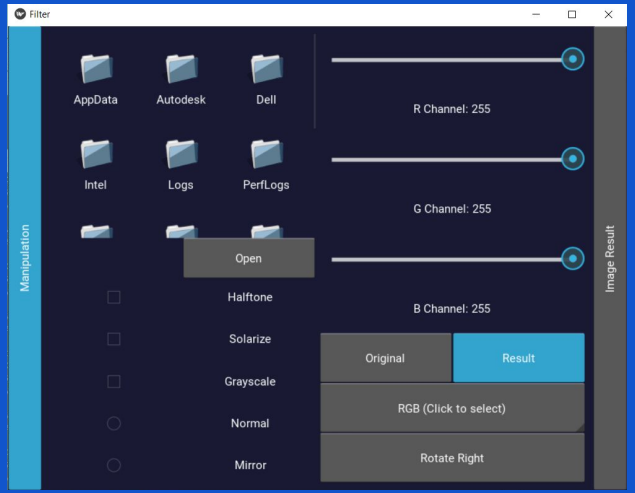
This book is bundled with two Python/Kv example files. One being a simple demonstration of the difference between Kv and Python and the other being a fleshed-out demo of the capabilities of the Kv framework and complex layouts. They should provide a good introduction to Kv and Python. Try and replicate them or use their code in a creative way!

To open these files, simply open your command shell and run:

```
py directory://helloworld.py # replace directory with your directory and helloworld with the name of the python file
```

It is not necessary to open the .Kv file.

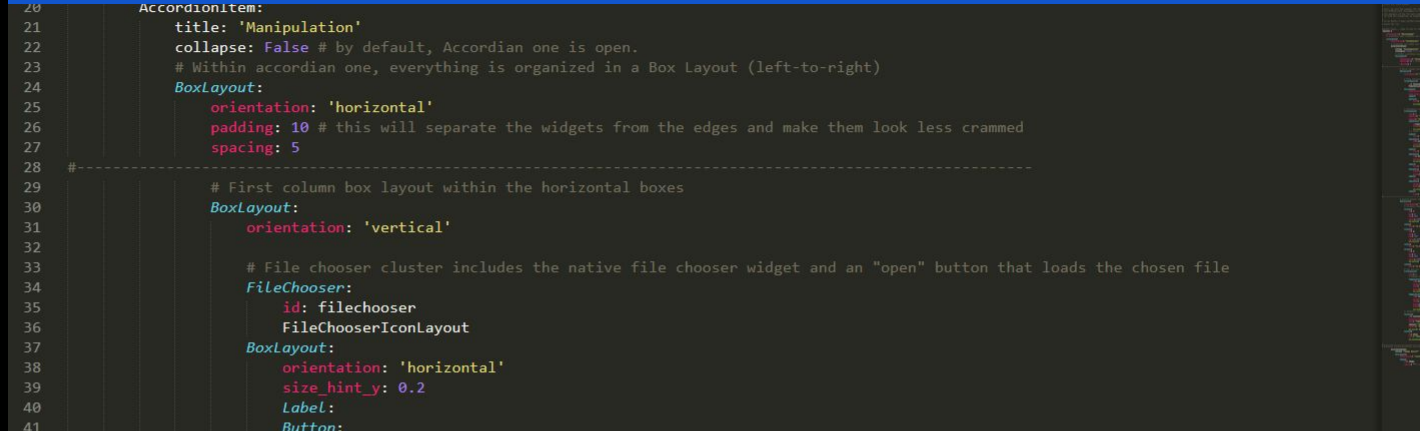




The example files

The best way to learn Kv is by trying it yourself! Make sure to use the comments provided in the example files. Here are a few pointers:

- Python is used for behavior (the back-end operations) and Kivy is used for appearance.
- This means that no Python functions should be defined in Kv and no Kv widgets should be defined in Python
- Use the setup quick reference to learn how to link your Kv and Python files effectively
- Each widget in the Kv should have an id for easy reference in Python, be part of a layout, and, if necessary, have a trigger for its behavior.
- Order matters in the layout! Make sure to be cognizant about how you organize your widgets.



Quick Reference: Common Properties and Behaviors

Properties (for more information on the Kv language, classes, and properties, visit https://kivy.org/doc/stable/guide/lang.html). These are used to define the look of a widget.

id	Assigns an ID to the widget that can be evaluated using Python (self.ids.[widgetname].[property])
text	Any text on the widget can be edited using this property.
orientation	Used in layouts and sliders, orientation defines the direction widgets are organized in.
state	Used in toggle buttons, and other booleans where state is binary.
value	Used in sliders, dropdowns, etc. where value is a “spectrum.”
size_hint_[dimension]	This can be used in a BoxLayout or GridLayout to define the relative size of a certain dimension. For example, “size_hint_y: 0.8” would mean that the widget takes up 80% of the y axis.

Quick Reference: Common Properties and Behaviors

Behaviors (for more information on the Kv language, classes, and properties, visit https://kivy.org/doc/stable/guide/lang.html). These are used to define how a widget behaves given certain actions.

on_release	Triggers on the release of a button. (typically used with buttons as opposed to on_click for improved UX)
on_active	Triggers when a checkbox is activated or deactivated. (Use ids.checkbox.active to make the distinction in Py)
on_value	Triggers when the value of a dropdown or slider changes.
on_text	Triggers when the selection on a dropdown changes. (Superior to on_value for dropdowns)
on_state	Triggers when the state of a toggle button changes.

3

Designing your UI

How to keep your user in the know.

Prioritizing User Experience

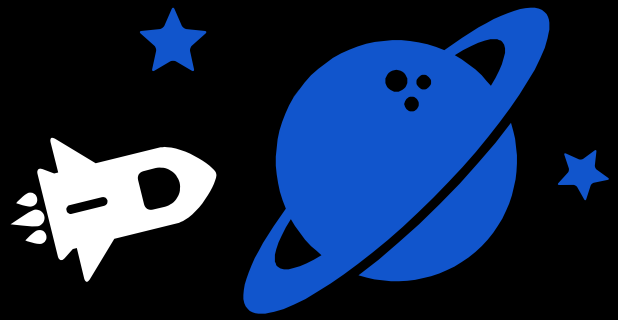
Here are a few tips on how to create a friendly user experience:

- ➔ Remember to keep the interface simple and clear. Hiding buttons and features only serves to confuse users.
- ➔ Make features recognizable and consistent. A standard textbox is white with a inner-edge drop shadow. Do not deviate from the standard, since it can cause your inputs to become unrecognizable.
- ➔ Plan the page layout. Always prototype with a flowchart on paper. Make sure interactions are logical and would make sense to an end-user.
- ➔ Use typography and color to create a clear hierarchy of importance. A red, bold button stands out more than a grey, italic button for example.
- ➔ Make sure to provide feedback. It can be jarring for users when they click something and nothing happens. Even if the program is running in the background, make sure to have some indication to the end-user.

Key Example: Yeco

Yeco is a good example of UX prioritization using Kivy. It is cleverly organized using BoxLayouts. The team goes into detail color coding related inputs, optimizing the size of each widget, and standardizing their layout. The menu bar remains at (as is tradition) the top of the window. Macro functions are on the right.

Overall, Yeco does a good job making Kivy look like its own while not sacrificing user experience.



4: Where to next?

Unlocking the full potential of Kv.

Crossing the Platforms

One of Kivy's greatest advantages is its cross platform versatility. As a result, many developers have used it to code apps in iOS, Android, Raspberry Pi, macOS, and Windows. It's easy and flexible when migrating and compiling to other platforms. To read more about compiling your code onto a separate device, please visit:
https://kivy.org/doc/stable/installation/installation-android.html

Now that you know how to use Kivy, you can easily and quickly prototype UIs for your Python programs. However, outside of this basic introduction, Kivy is capable of a lot more than just creating simple UIs. It can be used in visualization, scientific simulations, and a plethora of other applications to control variables and observe the different outcomes. However, this textbook will not have time to touch on those topics. It's time we left you to flourish on your own. Good luck!

Crossing the Platforms

Fin.

Bibliography

Sources and Further Reading:

https://kivy.org/doc/stable/guide/
https://stackoverflow.com/questions/tagged/kivy
http://www.yeco.io/