

You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)

# Building Android Apps With Python: Part -2

Step by Step Guide to Build Android Apps using Python



Kaustubh Gupta [Follow](#)  
Aug 11, 2020 · 6 min read ★

In the previous part, we discussed

- 1. What is Kivy and how to use it
- 2. Why Kivy is required
- 3. And built a basic app displaying “Hello world”

If you haven’t read the previous post then I highly recommend you to have a look at Part-1 so that you are familiar with the terminologies:

## Building Android Apps With Python: Part -1

Step by Step Guide to Build Android Apps using Python

[towardsdatascience.com](https://towardsdatascience.com)



In this second part, we will look at **various key elements of Kivy** which are the building blocks of any app.





Photo by [Priscilla Du Preez](#) on [Unsplash](#)

## String Builder

Before we proceed to various components of the app-building, I want to bring out a useful feature of integrating Kivy with Kivymd. Kivy has its own language that is dedicated to describing the user interface and various interactions. It is useful as the UI file can be separated from the logic file making the code more readable and manageable. We should separate these logics as in the .kv file, we don't have to make any import statements. We need not worry about where the specific element is located in the Kivymd directory, also we don't need to explicitly bind a widget to its root widget. KV files follow a hierarchical structure where you can define one root widget and whenever you define components below this, they are by default added to the root widget. We will look at this concept once again in MDLabel's example.

. . .

## Key Elements of KivyMD

We have talked enough about the basics, let's dive-in into the building blocks of app-building:

### Screen

All the things require a stage. The screen element is the first thing an app will have. All the activities, components are defined on a screen. It is defined simply by calling it in the KV file.

```
Screen:
    components
    .
    .
    .
```

### MDLabel

We have used this component several times during the introduction and demo part. This component makes it possible to display any text on your app screen. We can display any component in many ways:

```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3
4
5  class Main(MDApp):
6      def build(self):
7          return Builder.load_file("label.kv")
8
9
10 Main().run()
```

label.py hosted with ❤ by GitHub

[view raw](#)

Logic file

```
1  Screen:
2      MDLabel:
3          text: "[b]Thanks for reading my Medium article![/b]"
4          halign: "center"
5          font_style: "H4"
6          markup: True
```

label.kv hosted with ❤ by GitHub

[view raw](#)

Kivy Language file

Or the other way:

```
1  from kivymd.app import MDApp
2  from kivymd.uix.screen import Screen
3  from kivymd.uix.label import MDLabel
4
5
6  class Main(MDApp):
7      def build(self):
8          screen = Screen()
9          label = MDLabel(text="[b]Thanks for reading my Medium article![/b]",
10                          halign='center',
11                          font_style='H4',
12                          markup=True)
13          screen.add_widget(label)
14          return screen
15
16
17  Main().run()
```

label\_without\_kv.py hosted with ❤ by GitHub

[view raw](#)

Label without Kivy Builder

Both of them returns the same output:





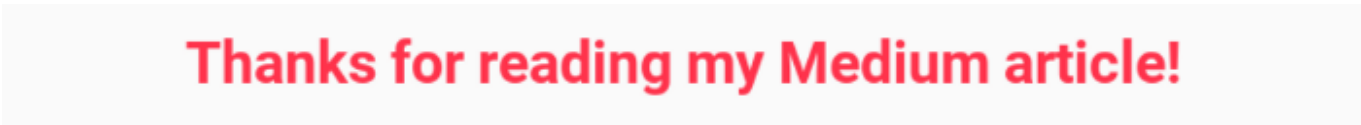
Example of MDLabel

MDLabel has various properties to customize:

- 1. **font\_style:** Available options are ‘H1’, ‘H2’, ‘H3’, ‘H4’, ‘H5’, ‘H6’, ‘Subtitle1’, ‘Subtitle2’, ‘Body1’, ‘Body2’, ‘Button’, ‘Caption’, ‘Overline’, ‘Icon’.
- 2. **text:** The text you want to display on the app.
- 3. **theme\_text\_color:** Available options are ‘Primary’, ‘Secondary’, ‘Hint’, ‘Error’, ‘Custom’, ‘ContrastParentBackground’.
- 4. **text\_color:** Text color is given in RGBA format. The values are given as percentage values and not absolute values. It means that the RGB values you have chosen must be divided by 255 to get the percentage values. Note that while using custom colors, you need to specify **theme\_text\_color to custom**. Consider this example:

```
text_color=(1, 0.2, 0.3, 1),
theme_text_color='Custom'
```

When this is applied to the existing example, you will get this result:



MDLabel with customized color

- 5. **halign and valign:** We can assign the position of the text horizontally and vertically. Available options are auto, left, center, right, and justify.
- 6. **markup:** We can use the markup language to customize the text. You need to specify markup = True to use the markup options. Check the kivy markup [documentation for all the available actions](#).

All the properties discussed here remain the same for most of the components except the ones that are exclusive for Labels. You can check more options in the [official documentation](#).

### MDButton

It enables you to make the UI interactive by binding the button action with other widgets. These bindings are performed by referring to each other ids. There are various types of buttons available in Kivymd and you can choose any button which suits your requirements. Available options are:

- MDIconButton

- MDFloatingActionButton
- MDFlatButton
- MDRaisedButton
- MDRectangleFlatButton
- MDRectangleFlatIconButton
- MDRoundFlatButton
- MDRoundFlatIconButton
- MDFillRoundFlatButton
- MDFillRoundFlatIconButton
- MDTextButton
- MDFloatingActionButtonSpeedDial

I have coded an example which covers the most frequent buttons used for better understanding:

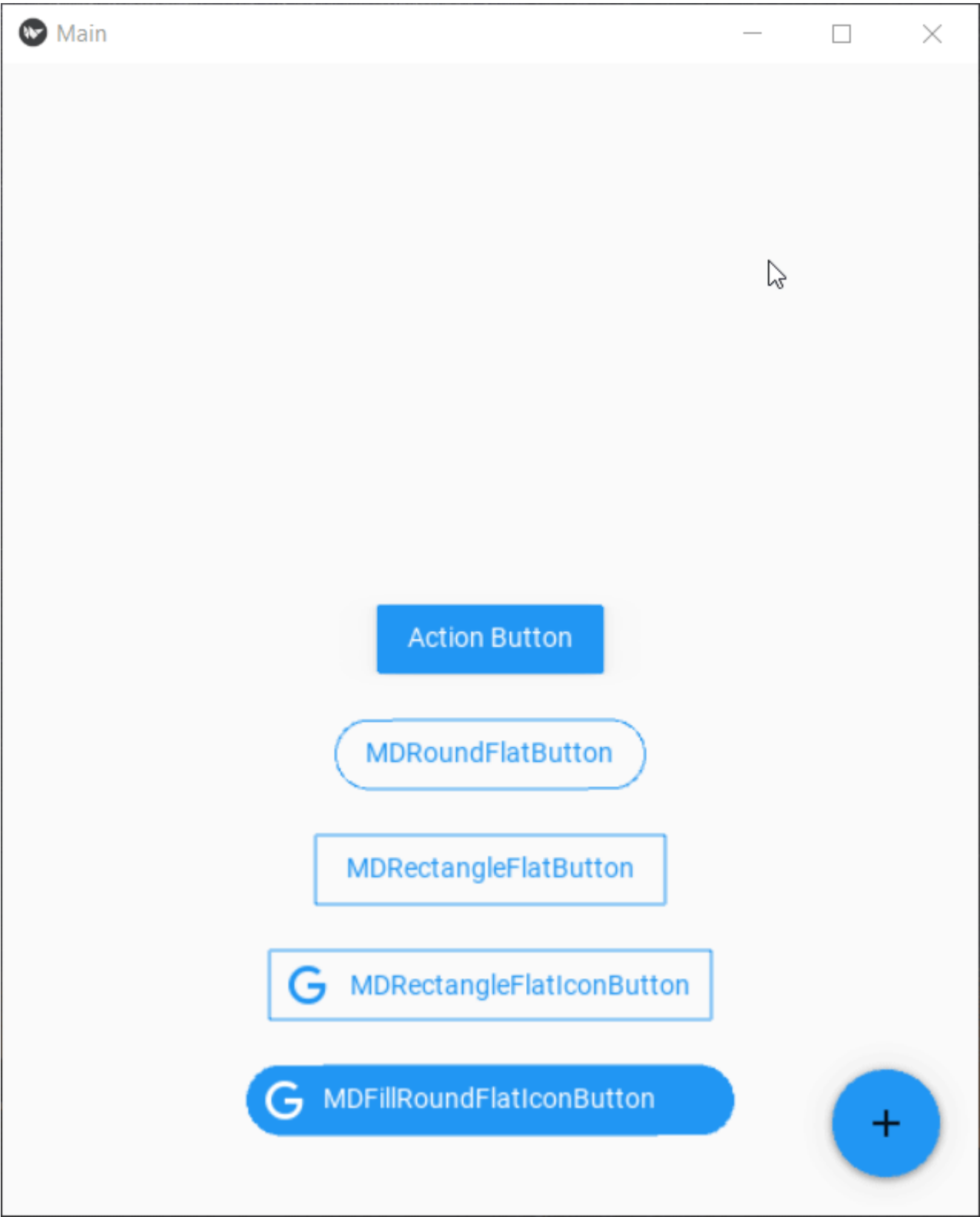
```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3
4  kv = """
5  Screen:
6      MDLabel:
7          text: ""
8          id: txt
9          pos_hint: {'center_x': 0.5, 'center_y': 0.6}
10     MDRaisedButton:
11         text: 'Action Button'
12         pos_hint: {'center_x': 0.5, 'center_y': 0.5}
13         on_press:
14             app.action()
15     MDRoundFlatButton:
16         text: 'MDRoundFlatButton'
17         pos_hint: {'center_x': 0.5, 'center_y': 0.4}
18     MDRectangleFlatButton:
19         text: 'MDRectangleFlatButton'
20         pos_hint: {'center_x': 0.5, 'center_y': 0.3}
21     MDRectangleFlatIconButton:
22         text: 'MDRectangleFlatIconButton'
23         pos_hint: {'center_x': 0.5, 'center_y': 0.2}
24         width: dp(230)
25         icon: 'google'
26     MDFillRoundFlatIconButton:
27         text: 'MDFillRoundFlatIconButton'
28         pos_hint: {'center_x': 0.5, 'center_y': 0.1}
29         width: dp(230)
30         icon: 'google'
31     MDFloatingActionButtonSpeedDial:
32         data: app.data
33         rotation_root_button: True
34     """
35
36
37  class Main(MDApp):
38      data = {
39          'language-python': 'Python',
40          'language-php': 'PHP',
41          'language-cpp': 'C++',
42      }
43
```

```
44     def action(self):
45         label = self.root.ids.txt
46         label.text = "This text is displayed after pressing button"
47
48     def build(self):
49         return Builder.load_string(kv)
50
51
52 Main().run()
```

buttons.py hosted with ❤ by GitHub view raw

Different Types of Buttons

Here I have separated the UI code in a variable and then loading that variable in the builder method. This is also a great approach if you don't want to make separate files and manage the whole code in one file. The output is:



GIF by Author

Let's understand what's going on:

1. Different buttons are displayed on the same screen.
2. Various icons can be included in buttons. Check out the [whole list of supported icons here](#).

3. The floating action button is something new here. It is widely used to give relevant social media links, share options, and many more actions depending on your creativity. It is getting the data to be displayed from the backend layer, our python code in the form of a dictionary where the key is the icon name and the value is text to be displayed. The app.data searches in the app (the root widget) for the data variable and the Main class of our app hold that variable. It means we should define these variables in the class it belongs to, you will get a better idea when we will build apps with multiple screens.
4. The action button is also new. The action which I have associated with this button can be done for any button type. Buttons have a lot of properties for every type but the most common of them is on\_release, on\_press, on\_close. As the name depicts they are triggered when that specific action occurs. Check the [documentation for all the actions](#).

MDTextField

Now we want to take inputs from the user, process it, and display the results. MDTextField makes it super easy to customize a base text input. It also comes in 3 classes MDTextField, MDTextFieldRound, and MDTextFieldRect. A text field has the input line, some light text (here known as hint text) to indicate what to input, some additional information below the line (here known as helper text), and sometimes an icon too. Let’s take an example:

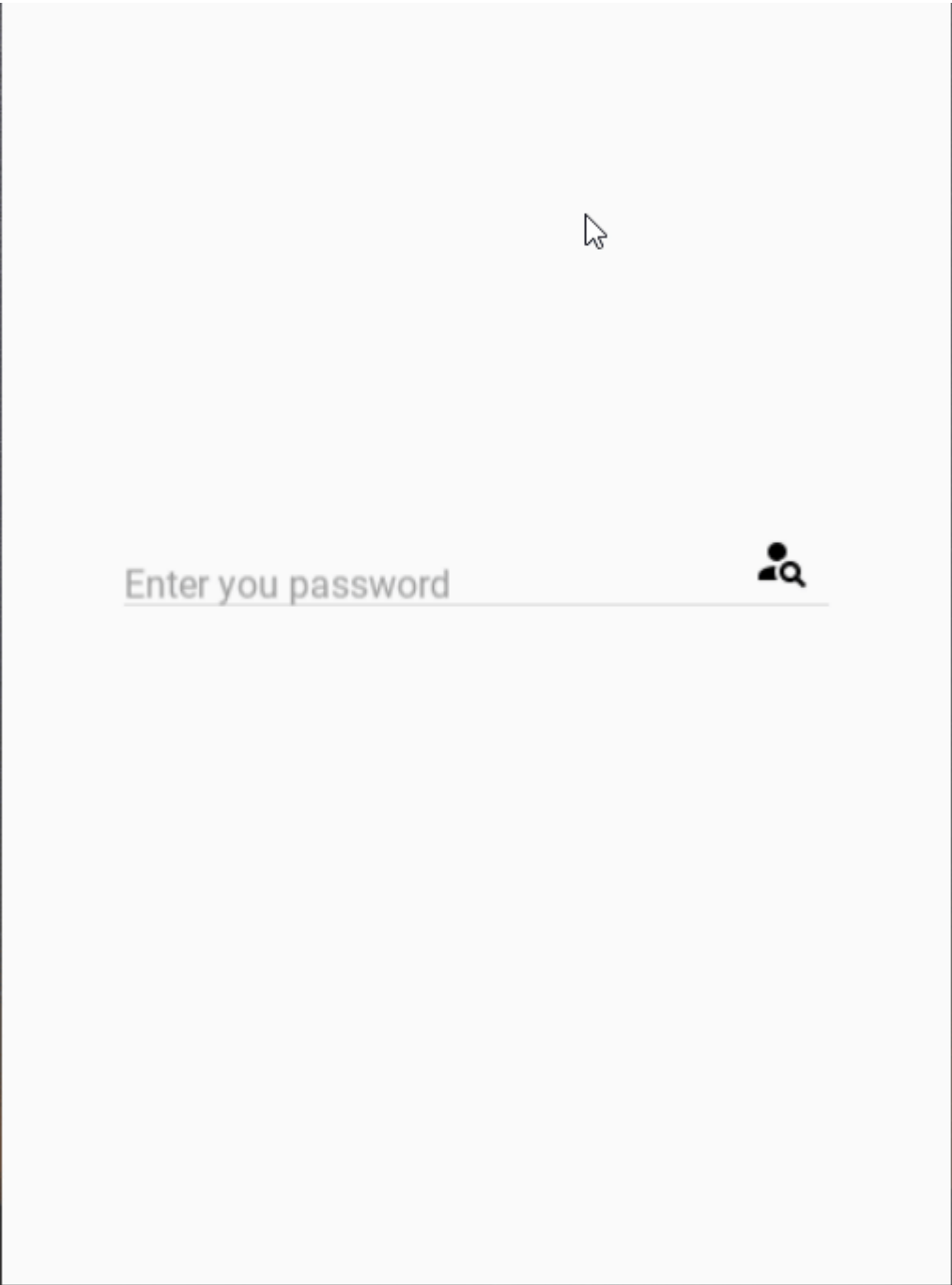
```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3
4  kv = """
5  Screen:
6      MDTextField:
7          hint_text: 'Enter you password'
8          helper_text: 'Forgot your password?'
9          helper_text_mode: "on_focus"
10         pos_hint: {'center_x': 0.5, 'center_y': 0.5}
11         size_hint_x: None
12         width: 300
13         icon_right: "account-search"
14         required: True
15
16  """
17
18
19  class Main(MDApp):
20      def build(self):
21          return Builder.load_string(kv)
22
23
24  Main().run()
```

textfield.py hosted with ❤ by GitHub view raw

Example of MDTextField







GIF by author

When the user taps on the field, the hint text automatically shifts on the top. Some useful properties of MDTextField are:

1. The helper text has various modes: “focus”, “persistent” and “error”. In “focus” mode, the helper text only appears when the text field is focused or tapped whereas, in “persistent” mode, the helper text always appears.
2. We can specify whether the field is required or not using `required=True`, control the length of text using `max_text_length`, or make it multi-line input.
3. The input can be taken to the python layer by defining an object property in the base class of the app and defining its reference in the KV file which references the id of the text field. See this example:

```
from kivy.properties import ObjectProperty
#In the KV part add this,
"""
Screen:
    variable_in_class: id_of_text_field

class Main(MDApp):
    variable_in_class = ObjectProperty(None)
    .
    .
    .
    # when you want to use that input text use:
    answer = self.root.variable_in_class.text
```

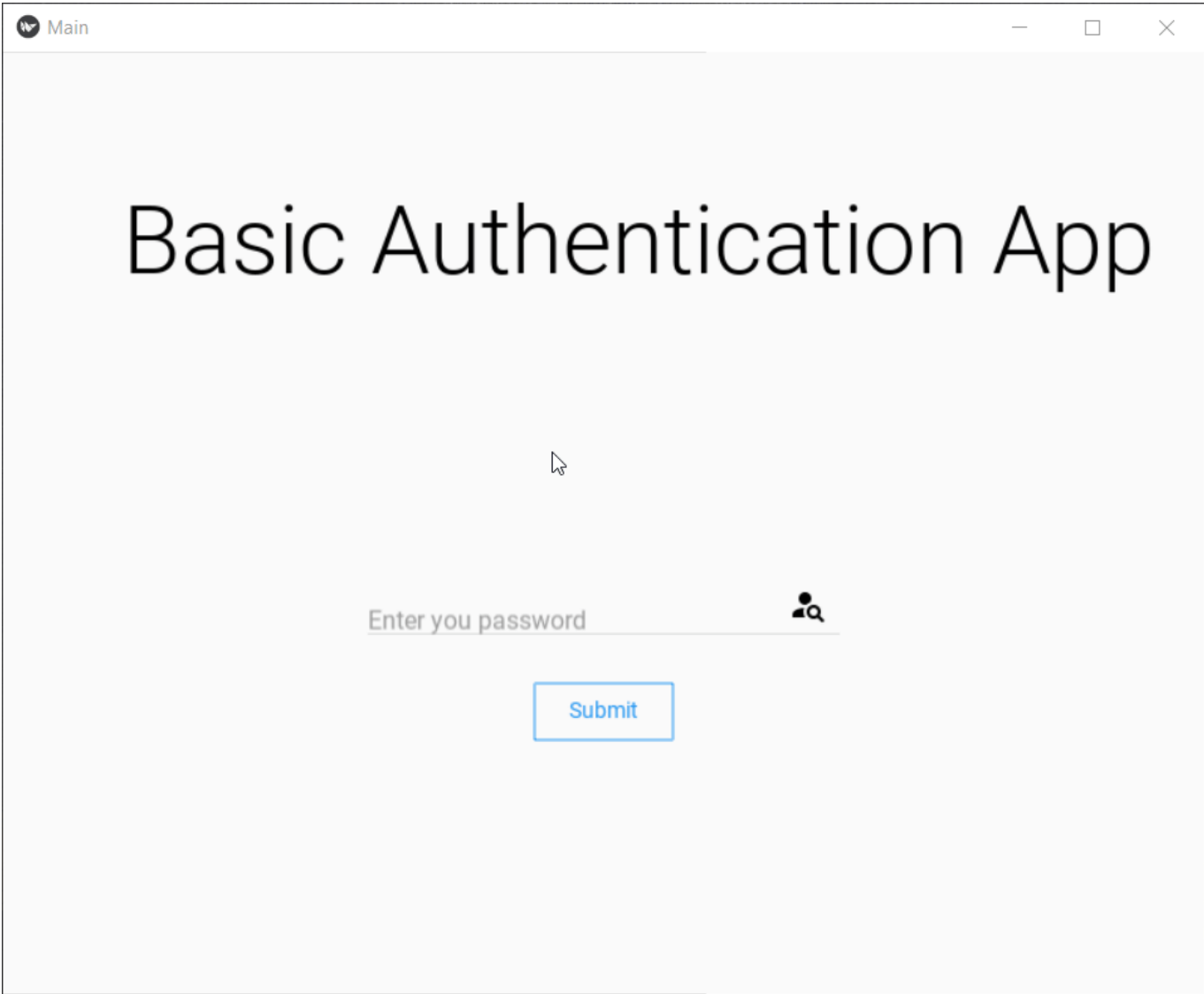


Check out [other properties here](#).

## Let's make a basic app!

We have covered only a portion of elements but before ending this article let’s combine what we have learned till now into a minimalistic app that takes password as input to display whether we are the root user or not. I would suggest you open up your IDE’s and try to build it yourself and compare it with my code!

```
1  from kivymd.app import MDApp
2  from kivy.lang import Builder
3  from kivy.properties import ObjectProperty
4
5  kv = """
6  Screen:
7      in_class: text
8      MDLabel:
9          text: 'Basic Authentication App'
10         font_style: 'H2'
11         pos_hint: {'center_x': 0.6, 'center_y': 0.8}
12     MDTextField:
13         id: text
14         hint_text: 'Enter you password'
15         helper_text: 'Forgot your password?'
16         helper_text_mode: "on_focus"
17         pos_hint: {'center_x': 0.5, 'center_y': 0.4}
18         size_hint_x: None
19         width: 300
20         icon_right: "account-search"
21         required: True
22
23     MDRectangleFlatButton:
24         text: 'Submit'
25         pos_hint: {'center_x': 0.5, 'center_y': 0.3}
26         on_press:
27             app.auth()
28
29     MDLabel:
30         text: ''
31         id: show
32         pos_hint: {'center_x': 1.0, 'center_y': 0.2}
33
34  """
35
36
37  class Main(MDApp):
38      in_class = ObjectProperty(None)
39
40      def build(self):
41          return Builder.load_string(kv)
42
43      def auth(self):
44          if self.root.in_class.text == 'root':
45              label = self.root.ids.show
46              label.text = "Sucess"
47          else:
48              label = self.root.ids.show
49              label.text = "Fail"
50
51
52  Main().run()
```



Combining what I have discussed so far

## Conclusion and what’s next

In this part, we covered the basic building blocks of Kivymd: Screen, MDLabel, MD Buttons, and MDTextField. In the next article, we will continue our journey and explore other complex and highly used components. If you liked this article, follow me on medium so you receive notifications about upcoming parts. With that said savonara!



**You can find me here:**

LinkedIn: [Link](#)

GitHub: [Link](#)

Update: Part-3 is here:

### Building Android Apps With Python: Part -3


Step by Step Guide to Build Android Apps using Python

towardsdatascience.com

### Sign up for Top 10 Stories

By The Startup

Get smarter at building your thing. Subscribe to receive The Startup's top 10 most read stories — delivered straight into your inbox, once a week. [Take a look.](#)

 Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

Python

Android

Kivy

Kivymd

Android App Development

Learn more.

Medium is an open platform where 170 million readers come to find insightful and dynamic thinking. Here, expert and undiscovered voices alike dive into the heart of any topic and bring new ideas to the surface. [Learn more](#)

Make Medium yours.

Follow the writers, publications, and topics that matter to you, and you'll see them on your homepage and in your inbox. [Explore](#)

Write a story on Medium.

If you have a story to tell, knowledge to share, or a perspective to offer — welcome home. It's easy and free to post your thinking on any topic. [Start a blog](#)



[About](#) [Write](#) [Help](#) [Legal](#)