

MOBILE APP DEVELOPMENT

With Python & Kivy



<a>Journey to becoming a digital developer

Musawenkosi Nyathi

Content Page

<u>Chapter one: Getting started with Python</u>	<u>4</u>
<u>1.1: Starting out</u>	<u>4</u>
<u>1.2: String data types</u>	<u>5</u>
<u>1.3: Set data types</u>	<u>5</u>
<u>1.4: Sets Exercise</u>	<u>7</u>
<u>1.5: Comments</u>	<u>8</u>
<u>1.6: List data types</u>	<u>8</u>
<u>1.7: Dictionary data types</u>	<u>10</u>
<u>1.8.A: Boolean & Mathematical operators</u>	<u>11</u>
<u>1.8.B: Integer & String addition</u>	<u>12</u>
<u>1.9: format() method</u>	<u>13</u>
<u>1.10: if & else statements</u>	

.....	13
1.11: User	
input.....	
.....	14
1.12: Chapter one	
exercises.....	
.....	15
1.13: Discussing chapter	
on.....	
.....	15

Chapter two: Core Python

.....	17
2.1: Dates &	
Time.....	
.....	17
2.2: Mathematics in Python.....	
.....	20
2.3: Random	
Module.....	
.....	22
2.4: split()	
method.....	
.....	23
2.5: for	
loops.....	
.....	24
2.6: while	
loops.....	
.....	25
2.7:	
Functions.....	
.....	26
2.8:	
Classes/Objects.....	
.....	27
2.9: Discussing Chapter	

two.....	28
2.10: Chapter 2 exercises.....	29
Chapter three: Python Projects.....	
.....	30
Project one: Basic calculator.....	
.....	30
Project two: A lottery program.....	
.....	33

Chapter four: Starting out with kivy

.....	35
4.1: installing and setting up kivy.....	
.....	35
4.2: first kivy app.....	
.....	37
4.3: customizing our app.....	
.....	39
4.4: planning your app.....	
.....	42
4.5: concept discussion of chapter four.....	43
4.6: properties in kivy.....	
.....	43
4.7: canvas.....	
.....	46
4.8: the kv language.....	

.....	46
4.9: touch methods.....	
.....	47
4.10: multi-touches.....	
.....	49
4.11: events.....	
.....	50
4.12: advanced graphics.....	
.....	53

Chapter five : creating mobile applications.....

.....	55
5.1: Widget manipulation.....	
.....	55
5.2: Pong game mobile app.....	
.....	59
5.3: Mobile calculator.....	
.....	70
5.4: Mobile clock & stopwatch.....	
.....	76

Chapter six: Introduction to SQLite3.....

.....	83
-------	----

Chapter seven : Complex mobile apps.....

.....	94
7.1: Brickly Math.....	

.....	94	
7.2: Hotel Management System.		1
.....	21	
7.3: Chat Server App..		
.....	150	
.....	Chapter eight: What's	
next.		
.....	218	

About The Author

Hi, nice to make your acquaintance :)

I am a self-taught app and web developer who has passion for computer programming and is currently pursuing a career in software engineering. To get where I am, I had to take the long road on learning, as there is so little information in app development with python out there. Due to lack of information in mobile app development with python, I decided to write this book to give all the knowledge I have in app development to all aspiring app developers in a very simple way. I love teaching other people how to code and it make me happy that you have decided to take the journey to digital development and the fourth industrial revolution will make computer programming an essential to all.

Computer programming is very easy to learn if you really put in the work to it. With experience and practice, you will eventually become a very good programmer.

Message from the author

To be able to understand everything that we will do in this book, it is important to copy what we do and not just read the book. After learning a new concept, close the book and rewrite the program off by heart at least 3 times, that way you will start to understand how it really works

Please feel free to give back feedback on how you are progressing. It does not have to be a high-class enterprise app; even the small projects will still make me proud. You can email the screenshot or videos of your projects. Do not hesitate to drop a suggestion or an error you found in the book.

All feedbacks should be emailed to **musanyathi20@gmail.com**
I hope this book will meet your needs. Enjoy! **this page was intentionally left blank**

Chapter One: Getting started with Python

Section 1.1: Starting out

Python is an expound world widely used programming language that is object orientated, created by Guido Van Rossum and was released in 1991. It has a large comprehensive standard library. In this book, we will only learn the Python fundamentals needed in Kivy in order to be able to build Applications. It is very easy to use and similar to the English language.

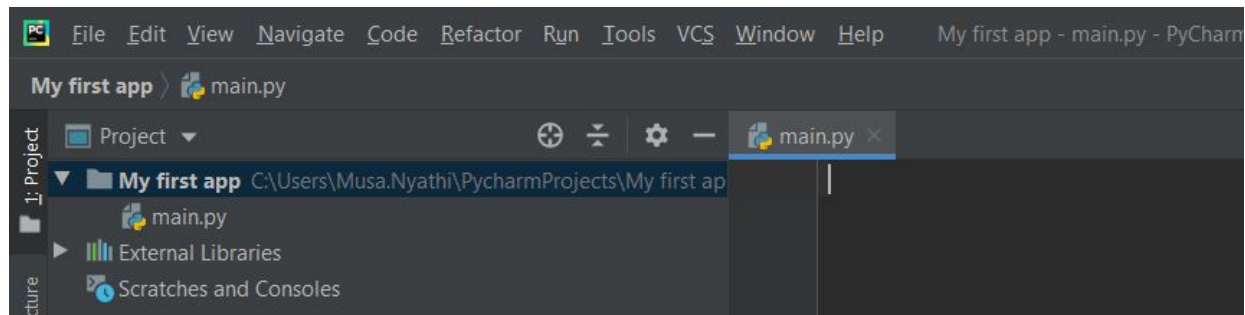
There are two versions currently being used, Python 2 and Python 3. In this book, we will be using python 3.7.7 throughout. You can download and install Python at www.python.org. To verify if python is installed, open your command prompt, and simply type in python and it would return something similar to the text below:

```
C:\Users\Musa.Nyathi>python
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 09:44:33) [MSC v.1900 32 bit (Intel)] on
win32 Type "help", "copyright", "credits" or "license" for more information.
>>>
```

This result shows us the current version of python in your computer that you downloaded (mine being **Python 3.7.7**).

Now that you have installed Python, you now need to install an Integrated Development Environment (IDE) where you will be writing your code and the IDE that we will be using in this book is **PyCharm**, which you can download at www.jetbrains.com.

After downloading and setting up your IDE, click on file, create a new project, and give it the title “**My first App**”. A file will be shown on the left hand side of you IDE with the title of the project, right click on it, create a new Python file, and name it “**main.py**”. You will have a setup similar to the below image:



Well done! You have created your first project and ready to get started.

Section 1.2: String data types

A string is a contagious set of characters represented in quotation marks. It can be either single or double quotation marks.

Think of a string as an item stored inside a box, e.g. if you have a box of books, the only way to know that there are books inside is if the box has a sticker outside labelled “books”, the books inside will be the strings and the label outside will be what we call a variable. **Variables** are elements or features that are liable to change.

Example:

```
books = “My Computer Science Books”  
print (books)
```

After typing in the code above, simply run it by right-clicking your main.py file and clicking run ‘main’ and it should print out the statement “My Computer Science books”.

Although we told the computer to print books, it instead printed our statement “My Computer Science books” and this is because “books” is our variable and “My Computer Science books” is our string. We store strings in variables so we can use them as many times as possible without having to keep typing the same strings over and over again as strings can be in big lists which we wouldn’t want to keep typing it up again. We will be looking at the different ways that strings can be stored in a variable over the next few sections.

Section 1.3: Set data types

Sets are unordered collections of unique objects that are stored inside curly brackets assigned to a variable and there are two type of sets: Sets and Frozen Sets. **Sets** are changeable and new elements can be added once the set is defined. **Frozen sets** are unchangeable once you define the set.

Below is an example of code that shows a Set:

```
universities = { "Harvard", "MIT", "Cambridge", "Oxford"}  
universities.add("Stanford")  
print (universities)
```

The above code shows us an example of a set. In line one we define our set under the variable "universities". In line two, we add a new item ("Stanford") into our set with the **.add** method, which therefore adds our item into the set and in line three; we call our variable with the **print** method, which returns our set with an added item.

Below is a sample code that shows a **frozenset**:

```
universities = frozenset({ "Harvard", "MIT", "Cambridge",  
"Oxford"}) print (universities)
```

The above code shows us an example of a frozenset. We define our set as a frozen set by putting it inside brackets of a frozenset element. This tells the computer to freeze that set and therefore you cannot add a new item or remove an item inside the set. If you try to add a new item inside the set, you will get an error like the text below:

Traceback (most recent call last):

```
File "C:/Users/Musa.Nyathi/PycharmProjects/My first  
app/main.py", line 3, in <module> universities.add("Stanford")  
AttributeError: 'frozenset' object has no attribute 'add'
```

This simply tells us that the frozenset method has no 'add' attribute, which means you cannot add a new item into a frozenset.

Removing an item in a set:

We remove items from a set(s) with the **.discard()** method and we can only remove items in sets and not frozen sets. Below is an example of the **.discard()** method:

```
universities = { "Harvard", "MIT", "Cambridge", "Oxford"}  
universities.discard ("Cambridge")  
print (universities)
```

This will let us remove "Cambridge" from our set and when we print this, it will return the three remaining universities remaining on our set.

Deleting an entire set:

We delete an entire set using the **del** method whereby we simply delete our variable and everything in it gets deleted too. Below is the sample code:

```
universities = { "Harvard", "MIT", "Cambridge", "Oxford"}  
del universities  
print (universities)
```

When we run this, it will return an error because we deleted our variable ("universities") in line 2 and then ran our program in line 3 which the variable had been deleted and will no longer be defined.

Joining sets:

We can have two or more sets in a program and we can always combine them to print them out as one using the **.union()** method . Let us look at the example below:

```
letters = { "a", "b", "c", "d"} numbers = {1, 2, 3, 4}  
both = letters.union (numbers) print(both)
```

Notice that we did not put our numbers in quotation marks; this is because numbers are either integers (numbers without decimals) or floats (numbers with decimals) and we only put strings (letters and words/names) in quotations unless they are attributes or words used by the program to perform an action.

Section 1.4: Sets exercises

1. Make a program of four sets of five of your friends under the variables: Names, Surnames, Ages, Countries.
 - a. Join the variables "Names" and "Ages" under a new variable (set3)
 - b. Join the variables "Surnames" and "Countries" under a new variable(set4)
 - c. Print set3 and set4 in the program
2. Create a set of six of your favorite fruits and another of six of your favorite colors.
 - a. Remove one fruit from your fruits set.
 - b. Delete your colors set
 - c. Print your fruits set.

Section 1.5: Comments

Comments are used as explanation of the code to make it more readable and understandable to an external person who might be given the code to work on or use too. Python ignores comments after they are declared with the symbol **#** for a single line comment. For comments in more than a single line they are declared using triple quotation marks and also closed by triple quotation marks to mark the end of the comment. The following are examples of comments:

```
#This is a single line comment """this is a  
multiline comment"""  
print ("Hello World!")
```

You can also write your comment next to your code and the program will only read the code and omit the comment.

Section 1.6: List data types

Lists are ordered collections of objects/items that allow duplicate members. Unlike a set, a list is stored inside square brackets []. By ordered, we mean that the list will be printed exactly the way you wrote it without any position change. Below is an example of a list and its returned result after being printed:

```
cars = ["BMW", "Ferrari", "Volkswagen", "Corolla", "Tesla"] print  
(cars)
```

```
"C:/Users/Musa.Nyathi/PycharmProjects/My first app/main.py"  
['BMW', 'Ferrari', 'Volkswagen', 'Corolla', 'Tesla'] Process finished  
with exit code 0
```

The first textbox shows our program and the second textbox shows our result. You can delete, add or join lists using the same methods we used in the set data types section.

Lengths, range and list substitution

We can find the length of our lists, sets, dictionaries, etc. using the **len()** method and below is an example of checking the length of our list:

```
cars = ["BMW", "Ferrari", "Volkswagen", "Corolla", "Tesla"] print  
(len(cars))
```

The **len()** method is useful when we now have thousands of items in our list and it saves us a lot of time than having to count the items one at a time.

Although we have five items in our above list, the positions of our items start from index zero to four (not one to five), meaning we access our first item ("BMW") by calling position zero. Below is an example:

```
cars = ["BMW", "Ferrari", "Volkswagen", "Corolla", "Tesla"] print  
(cars[0])
```

This will return only the first item ("BMW") which is said to be in index zero. If you do not know how many items you have and want to access the last item, you can do that using negative indexing.

print(cars[-1]) will return the last item in your list ("Tesla")

You can also access only the range of the first three items in your list using the example below:

```
cars = ["BMW", "Ferrari", "Volkswagen", "Corolla", "Tesla"]  
print(cars[0:3])
```

You can change the item in a list simply by assigning its index to the new value. An example is shown below:

```
cars = [ "BMW", "Ferrari", "Volkswagen", "Corolla", "Tesla"]  
cars[2] = "Audi"  
print (cars)
```

We change the second index ("Volkswagen") in the cars list and assign "Audi" to it, which then will be replaced by "Audi" when it is returned. This allows us to change an item in our lists anywhere in the program without having to go looking for the list again.

Section 1.7: Dictionary data types

Dictionaries consist of key value pairs and are stored inside curly brackets. We can say dictionaries are like sets, except every item is assigned to a key. Dictionaries help us in terms of storing sets that could be attached to their pair keys, e.g. a student's information.

Below is an example of a dictionary of a student's details:

```
student_one = { "Name": "Lian", "Age": 25, "Major": "Physics" }  
print(student_one)
```

You can access a certain item in a dictionary, below is an example:

```
student_one = { "Name": "Lian", "Age": 25, "Major": "Physics" }  
print(student_one["Age"])
```

This will return only the age on a student.

You can change a value in a dictionary by calling its key-item and assigning it to its new value. Below is an example:

```
student_one = { " Name": "Lian", "Age": 25, "Major": "Physics" }  
student_one["Name"] = "Michael" print (student_one)
```

You can delete a value in a dictionary using the del method that we looked at the previous section. Let us look at an example by deleting the "age" key-value in the image below:

```
student_one = {"Name": "Lian", "Age": 25, "Major": "Physics"}  
del student_one["Age"]  
print (student_one)
```

This will now return only the "name" and "Major" key-value with its print the variable student_one. We will look at sets, lists and dictionaries in depth when we build our first program in section three.

Section 1.8: Boolean & Mathematical Operators

Boolean operators are true or false statements based on variable conditions, e.g. ==, >, <, <=, ==, ==. In this section, we will look at examples of how and where Boolean statements can be used. Let us do a few simple programs on Boolean operators:

```
print (5>3)
```

If you simply print this, it will return the statement "True" since five is greater than three.

```
number_one = 6
```

```
number_two = 7
```

```
a = number_one >= number_two
```

```
b = number_two <= number_one
```

```
print (a)
```

```
print (b)
```

Both the results of the above code will return the statement "False" because neither of the conditions are true.

Mathematical operators are symbols we use in python for mathematical terms, e.g. addition (+), subtraction (-), multiplication (*), division (/), squares (**)

We use mathematical operators mostly in programs that may need mathematics like simple calculator applications, lottery applications, etc.

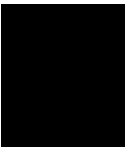
Let us look at the following images below with their results. We will not explain them, as it is basic mathematics:

```
number_one = 6  
number_two = 7  
a = number_one + number_two
```



Aprint (a)

"C:/Users/Musa.Nyathi/PycharmProjects/My first app/main.py" 13
Process finished with exit code 0



```
number_one = 6  
number_two = 7  
a = number_one * number_two Bprint (a)
```



"C:/Users/Musa.Nyathi/PycharmProjects/My first app/main.py" 42
Process finished with exit code 0

```
number_one = 6
number_two = 7
a = number_one ** number_two
```



```
C:\print (a)
```

C:/Users/Musa.Nyathi/PycharmProjects/My first app/main.py" 27993
Process finished with exit code 0

Section 1.8: Integer & String addition

A string and a number cannot be added . To make this possible you have to convert your number to a string. Let us look at a sample program below that shows the possibility of strings and number addition:

```
age = 24
print("I am " + str(age) + " years old.")
```

In the above code we have declared our age as a number, but printed it out as a string by assigning the **str ()** method on the print method which converts our integer to a string. If you had just called the age variable without assigning it to the **str()** method, it would have given an error.

Section 1.9: format() method

Although the program of our previous image works, we can make the code neater and easier to change without having to convert our

number (assigned to our age variable) to a string using the **format()** method, which we display in the sample program below:

```
age = 24 print("I am {} years old.".format(age))
```

The curly brackets are what hold out what will need to go into the printed sentence that will be taken from what is contained inside the **format ()** method brackets, which can be anything from a number to a word.

You should know that it is impossible to convert a string to a number.

Section 1.10: if & else statements

if & else statements are conditional statements that tell the program that if a certain condition is met it should return a certain results, else return a certain alternate result. These statements are used almost everywhere as almost every application in the world has multiple outcomes in their operations and these statements help us tell the computer what to return.

Below is an example of a program that consist of if & else statements:

```
number_one = 8
number_two = 6
If number_one < number_two:
print ("eight is greater than six")
else:
print ("six is not greater than eight")
```

The returned result will depend on our condition. In this case, we have made a condition that number one is greater than number two, which is untrue, and therefore we should get the second statement as our result.

if & else statements usually go together Boolean operators as both are conditional statements which compare certain objects. We can have more than two conditions in our program and therefore

we have more than two **if**- statements and this is where **elif** statements get introduced into our program, which means else-if. Let us look at the code below:

```
number_one = 8
number_two = 6
if number_one < number_two:
print ("eight is greater than six")
elif number_two == number_one:
print ("eight is equal to six")
else:
print ("six is not greater than eight")
```

In line five, we compare the two variables with double equal signs (**==**). This tells the computer that we are comparing the two variables and not assigning one to the other. The opposite of this is an exclamation mark followed with an equal sign (**!=**) which means "not equal to".

in the third chapter, we will be building our first python app and a few other apps and that is where we will go in depth on how **if** & **else** statements together with everything we have covered together can combine to build an app.

Section 1.11: User input

Input method

An input method is a method that allows the user to enter an input, and then an output returns with the conditioned result. Let us take an example where we ask for the user to enter their name and return a sentence telling them their name. Below is the example of code:

```
user_name = input("What is your name?") print ("your name is " + user_name)
```

The above code asks the user to input their name and returns the string "your name is" added with the user input.

Every input that a user enters is read as a string by the computer, therefore when working with numbers in user inputs, you have to tell the computer that the input will be an integer or float. Below is a sample code:

```
user_age = int(input ("How old are you?")) print ("you are {} years old".format(user_age))
```

We tell the program that our input is an integer since age cannot have a decimal in years and then use the format method to include our age in the sentence as an integer cannot be added to a string.

Section 1.12: Chapter 1 exercises

1. Create a variable called age and assign your age to the variable. E.g. Age=24 a. With your mathematical knowledge print out your age as the number of days that you have lived for. Ignore the leap year and assume every year has 365 days. b. Now print out your age as the number of seconds you have lived for in a sentence format. You have to tell the user that they have lived for xxx years.

2. Create a variable (birth_year) & assign your birth year to the variable. E.g. birth_year = 1997 a. Ask the user to input their birth year and your program return a sentence that tells them that you are either xxx years older than they are or xxx years younger than they are.

Hint: *Use if & else statements, mathematical operators and the format method.* b. Now do the same as the above exercise, but now tell the user the age difference in seconds.

3. Create a program that asks the user to input their first number and then asks them to input their second number and returns the sum of the two numbers.

4. Create a dictionary with any five details of a student that is still at school like their grade, age, etc. then return only the value assigned to the third index/position.

5. Create a set with five of your favorite songs and then add a new song into the set without directly editing your set and print out that set now having to consist of six songs.

Section 1.13: Discussing Chapter 1

I would like to start by saying congratulations; you have built your first python programs. By now you should have noticed a few things that you might have struggled with in your code and if you had decided to give up, the following might have been the mistakes you might have had:

Variable names: The names of variables can never have more than one word (no spaces) and the only way to make this possible is by using the underscore symbol to join the words. E.g. `my_variable = "item"`.

Case sensitivity: Python is case sensitive and if you name your variable starting with a capital letter and call it later starting with a small letter, it gives an error. In Boolean operators, the `true` and `false` must always start with capital letter in order to work.

Indentation: for any method that ends with a colon, the following line must be indented inside it, either by a tab or by four spaces front. Pycharm usually does this automatically for you.

Chapter Two: Core Python

In the previous chapter, we have learnt the basics of python that has taught us to build very simple applications. In this chapter, we will look at some advanced python that we will need to start building more complex apps and that will give us knowledge and prepare us for the next chapter where we will be building projects on python before diving into Kivy.

I would advise you to type every code into your IDE and not just read the book as doing helps with understanding. More understanding will come from the experience in chapter three when we start to build fully functional python programs with everything that we would have learnt in chapter one and two.

Section 2.1: Dates & Time

Dates and time are useful in many real world applications like calendars, stopwatches, chess clocks, etc. In this section, we will look at the different methods of using dates and time.

To include dates and times in our programs, we first to **import** the **datetime** module. Modules are external sets of functions that we import in order to use them. They are functions that already exist and you just need to import them like the **datetime** module. This saves us a lot of time to be building complex functions that already exist. We will cover modules in detail later in this chapter.

Example: let us import the datetime module and display the current date.

```
import datetime  
todays_date = datetime.datetime.today()  
print (todays_date)
```

This will return today's date together with the time in the format (yyyy-mm-dd hh:mm:ss:ms) which contains the year-month-day , hour: minute: second: microsecond.

You may sometimes want to print only the year or only the day or time. To do this, you will need to specify to your program what exactly you want to be executed by the program. Let us look at the following example on what specifics are there on python.

Executing only the year:

```
import datetime today_date = datetime.datetime.today() print  
(today_date.year)
```

Executing only the day:

```
import datetime today_date = datetime.datetime.today() print  
(today_date.day)
```

Executing only the time:

```
import datetime today_date = datetime.datetime.today() print  
(today_date.time())
```

Executing only the hour:

```
import datetime today_date = datetime.datetime.today() print  
(today_date.hour)
```

There are many other conditional executions like only the seconds, minutes or microseconds and it will always depend on what you want to display on your program. You can also change the format of your dates into strings that can be read using the **strftime()** method. We insert certain keys in the brackets with a percentage sign just before it to indicate what format we need it on. Example:

```
import datetime today_date = datetime.datetime.today() print  
(today_date.strftime("%Y"))
```


“% Y” is a directive, the capital letter “Y” tells the program to execute the year in its full version, and the lower case “y” tells the program to execute the year in its short version. Below is a table of other directives.

Operator Description example

%a Short version of weekday Sat
%A Full version of weekday Saturday
%b Short version of month Jun
%B Full version of month June
%f Microsecond 375261
%H Hour 19
%M Minute 18
%S Second 57

There are a lot more operators that exist in python, but the above are the mostly used and are fundamental to any apps that might require you to work with dates and time. You do not need to try to memorize these all by head, just have an understanding on how they work and how they are used and in what requirement.

Section 2.2: Mathematics in Python

Mathematical functions are defined in the maths module, so we can always import and use them instead of recreating them, as they can be sometimes complex and time consuming. These include logarithmic functions, trigonometrical functions, conversion functions, power functions, representation function and many more. Not all mathematical functions are defined in the math modules; some are already built in just like the addition, subtraction, etc. I believe we have covered all the built in mathematical functions that are fundamental except 3 which are the minimum, maximum and absolute functions, which we will be covering up next.

Example of the min() function:

```
numbers = {2, 5, 9, 16, 4} print(min(numbers))
```

The above program will then execute the minimum number in our

numbers set which would be the number 2

Example of the max() function:

```
numbers = {2, 5, 9, 16, 4} print(max(numbers))
```

The above program will then execute the maximum number in our numbers set which would be the number 16

Example of the abs() function:

```
x = 18 print(abs(x))
```

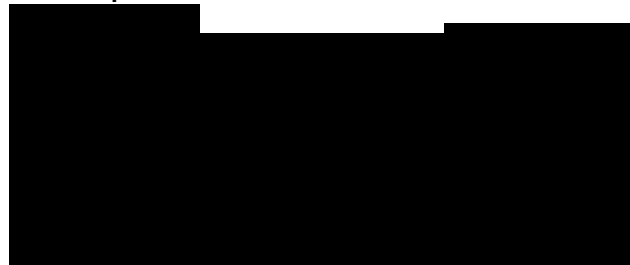
The above program will then execute the absolute value of -18, which will be executed as a positive number.

The math module

To use more complex math functions, we need to use the math module. To use the math module, you need to **import math**. There are various math functions, and in this section we will only show you a few examples just to get the understanding on how they can be used.

By now you should have realized that code can be written in many different ways and still return the same outcome.

Example:



```
import math x = 30  
y = math.cos(x) print(y)
```

```
import math  
x = 30  
print(math.cos(x))
```

```
import math  
print(math.cos(30))
```

All three-sample codes in the above example are written in three

different ways, but the executed results will remain the same in each sample code.

We can also use the math module to find constants such as pi (3.14), e (2.72) and radians of values. Examples of which functions can be used are shown below:

```
import math print(math.pi)
import math print(math.e)
import math
x = math.exp(10)
```

Section 2.3: Random Module

The random module is an import that allows us to generate random numbers in our programs. Random numbers are usually used in guessing games like magic number games. They are also used by online gambling companies like the lottery companies In order to generate random numbers. This is why it is difficult to win the lottery. To generate random numbers, you need to **import random**. There are two types of random numbers: **random integers** and **random floats**

Random integers

Example one:

```
import random x = random.randint(1,20) print(x)
```

In the above code, we declared a variable (x) as a random integer in the range 1 to 20. If you run the code, it will execute a random number and if you run the code again, there are very slim chances that the number might be the same, as the program will generate another random number.

Example two:

Let us create a program that generates a random number between 1 and 5 and then asks the user to enter a number and compares the two numbers and checks if they matched.

```
import random
magic_number = random.randint(1,5)
user_number = int(input("Enter a number between 1 & 5"))
if user_number == magic_number: print("you matched the magic number")
elif user_number > magic_number:
print("your number is greater than the magic number")
else:
print("your number is less than the magic number")
```

The code in the previous page is a magic number program that generates a random number in the range 1 to 5 and then asks the user to also input a number between 1 and 5. The program then compares the two numbers and returns a statement based on the conditions given. We gave our program two conditions and an alternate result if the two conditions are not met.

By now you should be getting the idea behind if & else statements, because those are the most used conditional statements in programs. You should also now be having a good understanding that everything we have done so far can be gathered into one source code in order to achieve a certain program.

Random floats

Example:

```
import random
x = random.random()
print(x)
```

The above code will execute a random float and if you run the app again it will execute another random float, making chances very slim to be the same as the first random number.

Note that the random floats method does not take any arguments, so if you put a range inside your brackets it will return an error message.

Section 2.4: the split() method

The `split()` method is used to split strings into lists where each word becomes an item and used for splitting numbers that are inputted by a user separated by commas, space and other things too. An example could be a scenario like the magic number program we did in section 2.3, but now we ask the user to enter two or more numbers and we want to split them.

Let us look at an example:

```
user_numbers = input("Enter your 3 numbers separated by  
commas") x = user_numbers.split(",")
```

This will split the numbers and return them as a list so that they are now individual items. Now we will be able to now compare the individual numbers to our magic numbers and see how many numbers we might have matched and won. We will look at how this method is applied in the real world when we build a lotto program in the next chapter.

Section 2.5: for loops

for loops are used when you have a block of code which you want to be repeated a couple number of times. They iterate over a sequence that could be a list, set, dictionary or string. This method lets us execute statements once for each item in a set, list or string.

Example:

```
numbers = [1, 5, 8, 9] for number in numbers: print(number)
```

This code will then execute our numbers individually, each being on a different line. In other IDEs they might still be executed in the same line, but without being separated by commas anymore.

We can also use **for loops** in when we want to print only a number or numbers in a certain range of numbers using the **range** function.

Below is an example:

```
for numbers in range(1,5): print(numbers)
```

This code will execute the number 1 to 4 individually.

We can also loop through strings, meaning we can individually print out letter individually in a word. Example:

```
for x in Ferrari: print(x)
```

This will individually print the letter from our word (Ferrari) individually. We can also have **else** statements in loops, which lets us have a statement printed out at the end of our loop.

Example:

```
for numbers in range(1,5):  
print(numbers)  
else:  
print("That's all the numbers")
```

This code will return the numbers 1 to 4 and then a statement at the end ("That's all the numbers").

Section 2.6: While Loops

While loops keep a block of code repeating until a Boolean condition is met.

Example one:

```
number = 500  
while True:  
  print(number)  
  number += 1  
if number >= 50000: break
```

The code will execute the number 500 and keep increasing by 1 until it reaches 50000, then breaks. If you do not include the last two lines, the number will keep increasing indefinitely.

Section 2.7: Functions

A function is a block of code, which performs specific tasks and only runs when it is called. A **def** statement defines it and in it, you can insert parameters.

Example:

```
def my_function(): print("This is my function.")
```

This will no return anything because you have only created your function, but have not called it.

To execute your function you need to call it together with its paranthesis:

```
def my_function(): print("This is my function.")  
my_funtion()
```

Let us now create a function at our magic_game program:

```
1. Import random  
2. magic_numbers = [random.randint(1,20), random.randint(1,20)]  
3.  
4. def ask_user_for_number():  
5. user_number = int(input("Enter a number between 1 & 20"))  
6. if user_number in magic_numbers:  
7. print("You won")  
8. else:  
9. print("You lost")  
10.  
11. ask_user_for_number()
```

In line two, we create 2 random numbers between 1 and 20. In line 4, we define our function and we later call it back in line 11, which then executes it. Everything in the function could be anything you want to be included in your program. You can have as many functions as you desire in your code. When having more than one function, you need to **return** you function in order to exit the function once done. A return statement with no arguments is the same as return none.

Example:

```
1. Import random  
2. magic_numbers = [random.randint(1,20), random.randint(1,20)]  
3.  
4. def ask_user_for_number():  
5. user_number = int(input("Enter a number between 1 & 20"))  
6. if user_number in magic_numbers:
```

```

7. print("You won")
8. else:
9. print("You lost")
10.
11. return user_number
12. ask_user_for_number()

```

Section 2.8: Classes/Objects

Classes provide all the standard features of object-oriented programming. Since python is object orientated, almost everything is an object. A class is an object constructor. Functions are stored inside classes.

Classes will probably be the hardest section you will go through, so it is recommended you practise a lot on them and create as many as you can from your imagination.

Example:

```

class My_Class_Marks:
def __init__(self,math,physics,chemistry): self.math = math
self.physics = physics
self.chemistry = chemistry

```

```

Michael = My_Class_Marks( 64, 93, 71)
Lindy = My_Class_Marks(72,77, 56)
Wendy = My_Class_Marks(97,100,69)

```

print (Michael.math) We can create a class for a school and in it we have put in a function that contains the math, physics and chemistry marks. The **self.math** and other subjects is what pulls out the arguments/parameters inside our function. The **__init__** is a constructor that is called when an object is created from a class and allows the class to initialize the attributes of the class. We then printed Michael's math mark and if we wanted to print all his marks we could have done it the following way:

```

print(Michel.math, Michael.physics, Michael.chemistry)

```

Example 2:


```

class Person:
def __init__(self, name,surname, age): self.name = name
self.surname= surname
self.age = age

person_one = Person("Musa","Nyathi", 24)
print("I am {} {} and i am {}".format(person_one.name,
person_one.surname, person_one.age)
)

```

Now we want to print out everything in a statement so we use the format method and separate all our replacement holders by commas since we had put 3 sets of curly brackets we also put 3 replacement functions inside the format method.

Section 2.9: Discussing Chapter two

Congratulations! You have covered the python section of this book and ready to move into the kivy section where we start building the real mobile apps that can be put into play store. You have probably faced many difficulties with the previous sections and might have found some concepts difficult, do not worry, it gets easier with practice and as you go on deeper into the book. Remember that you can only connect dots looking back and not forward. I advise you do every exercise in this book from this point forward. This will increase your understanding. The solutions to every problem is found at the last pages of the book and if you had failed an exercise, look at the solution, close the book and try again and write the code at least 3 times in order to remember and understand the code well.

Section 2.10: Chapter 2 exercises

1. Write a program and ask the user to enter a number. The number should be between 1 and 10. If the user enters a valid number display "Valid". Otherwise, display "invalid". (This logic is used in a lot of apps where values entered into input boxes need to be validated).

2. Write a program which takes two numbers from the user inputs and displays the maximum of the two.
3. Write a program and ask the user to enter the width and height of an image. Then tell the user if the image is landscape or portrait.
4. Your job is to write a program for a speed camera, for simplicity, ignore the details such as camera, sensors, etc. and focus purely on the logic. Write a program that asks the user to enter the speed limit. Once set, and then ask the user to enter a value less than the speed limit and the program should display "Speed okay". If the value is above the speed limit, the program should calculate the number of demerit points. For every 5km, one demerit point should be incurred and should be displayed. If the number of the demerit point is above 12, the program should display "License suspended".
5. Write a function called **bee_ware** that takes a number
 - a. If the number is divisible by three, it should return "bee".
 - b. If it is divisible by five, it should return "ware".
 - c. If it is divisible by both three and five, it should return "beeware".
 - d. Otherwise, it should return the same number.

Chapter Three: Python Projects

By now, you should have a good idea of how Python 3 works. If you have done the exercises and revised all your mistakes, you will now be able to see how simple coding can be with practice and experience. We will now take everything we have learnt in chapter one and two and build full python programs and after this, you should be able to do almost anything in all we have learnt. For every project, we will start by showing the whole code and then going through with line by line and after all the projects you will be able to build any project you want. Although we have covered all the fundamentals in the previous sections, there will be some new concepts through the projects that will be alternatives to other methods you have learnt already just in case you find them better. Please first copy the code into your learning machine and then go through the systematic phase already having your code on front of you.

Project one: A Basic Calculator

We will create our basic calculator with the following outcome desires:

1. We ask the user which operation they would like the program to perform (addition, multiplication, subtraction or division)
2. We ask the users to input their first value
3. We ask the users to input their second value
4. If the values are valid, the program must perform the calculation, otherwise display "invalid input"
5. Display the answer of the calculation to the user

```
def add(num1, num2): # adding 2 numbers (function) return num1 + num2
```

```
def subtract(num1, num2): # subtracting 2 numbers (function) return num1 - num2
```

```
def multiply(num1, num2): # multiplying 2 numbers (function) return num1 * num2
```

```
def divide(num1, num2): # dividing 2 numbers (function) return num1 / num2
```

```
print("Please select an operator -\n" \ "1. Add\n" \ "2. Subtract\n" \ "3. Multiply\n" \ "4. Divide\n")
```

```
select = float(input("Select operations form 1, 2, 3, 4 :")) #input from user as float
```

```
number_1 = float(input("Enter your first number: ")) number_2 = float(input("Enter your second number: "))
```

```
if select == 1:
```

```
print(number_1, "+", number_2, "=", add(number_1, number_2)) elif select == 2:
```

```
print(number_1, "-", number_2, "=", subtract(number_1, number_2))
```

```
elif select == 3:
```

```
print(number_1, "*", number_2, "=", multiply(number_1, number_2))
```

```
elif select == 4:
```

```
print(number_1, "/", number_2, "=", divide(number_1, number_2))
```

else:

print("Invalid input")

In line one to eleven is where we created our functions for our operators (addition, multiplication, division and subtraction) and returned what we want each operator to do:

def add(num1, num2): # adding 2 numbers (function) return num1 + num2

def subtract(num1, num2): # subtracting 2 numbers (function) return num1 - num2

def multiply(num1, num2): # multiplying 2 numbers (function) return num1 * num2

def divide(num1, num2): # dividing 2 numbers (function) return num1 / num2

In line 13 to 17, we display a text that shows our reader which choices he will have to input depending on what operation he wants done:

print("Please select an operator -\n" \ "1. Add\n" \ "2. Subtract\n" \ "3. Multiply\n" \ "4. Divide\n")

In line 22, we now let the user enter his option of which operator he wants, then let him enter his first number, then his second:

select = float(input("Select operations form 1, 2, 3, 4 :")) #input from user as float

number_1 = float(input("Enter your first number: ")) number_2 = float(input("Enter your second number: "))

The rest of the code is where we now use our conditional statements to tell the program what to execute if what operator is chosen. We use if, elif & else statements to set our conditions:

if select == 1:

print(number_1, "+", number_2, "=", add(number_1, number_2))

The rest of the code is not displayed as it is based on conditionals and has already been explained. Although we have built our basic calculator app, it is still not complete as a user might want to input more than two numbers for his calculation and might want different

types of operators in one calculation. For this, we will use an alternative method, which takes all the code we have written above into just two lines of code letting us calculate anything in our basic mathematics and use different operators simultaneously using the **eval** method.

```
calc = input("Enter calculation:\n")  
print("Answer: " + str(eval(calc)))
```

With just these two lines of code, you have created your full basic calculator which can take in any calculation and return its answer. I would still advise you to use the first method to create your calculator so you gain experience on using the functions, conditional statements, etc.

Project two: A Lottery Program

We will create our basic calculator with the following outcome desires:

1. Ask the user to enter six numbers separated by commas.
2. split the numbers into individual items
3. make the program generate random numbers
4. compare the user number to the random numbers
5. Print how many numbers the user matched and let the user win an amount of 100 to the power of the number of matched numbers, e.g. $100^{**3} = \$ 100\ 000$ won. **import random** **def menu():**

```
user_numbers = get_user_numbers()  
lottery_numbers = create_lottery_numbers()  
matched_numbers = user_numbers.intersection(lottery_numbers)  
print("You matched {}. You won ${}!".format(matched_numbers, 100  
** len(matched_numbers)))
```

```
def get_user_numbers():  
    number_input = input("Enter your 6 numbers between 1 & 25,  
    separated by commas: ")  
    number_list = number_input.split(",")  
    integer_set = {int(number) for number in number_list}  
    return integer_set
```

```
def create_lottery_numbers():  
    values = set()  
    while len(values) < 6:  
  
    values.add(random.randint(1,25)) return values  
menu()
```

Chapter Four: Starting off with Kivy

Kivy is a free and open source Python library for developing mobile apps and other multi-touch application software with a natural user interface. Kivy is one of the best frameworks to use as it can run in multi operating systems including Android, IOS, Linux, Windows, OS X, etc.

Kivy is very easy to use and straight forward, installing it and setting it up maybe the only most challenging part, but we will be going through a systematic process in this chapter to do it.

Section 4.1: Installing and setting up Kivy

You should already be having your python and an IDE (preferably PyCharm) on your PC so we will not be going over that installation phase again. If you have not installed the above mentions, please refer to chapter one in the first section. Ensure that you have internet connection for the following installations and set up.

For the installation, please open you command prompt on your windows apps and type in the following and wait for it to finish the installation before going onto the next:

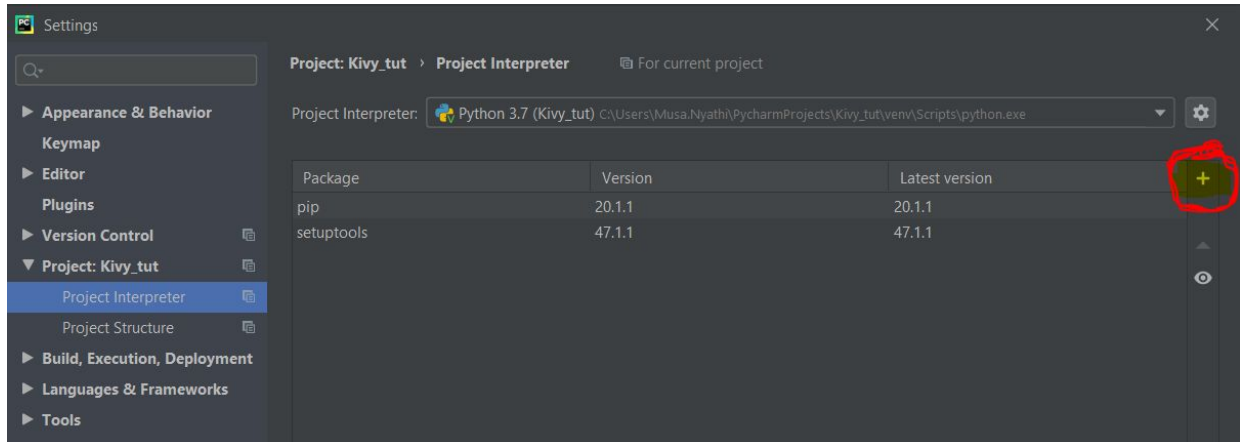
```
pip install cython
python -m virtualenv kivy_venv
kivy_venv\Scripts\activate
pip install kivy_deps.gstreamer==0.1.*
pip install kivy_deps.angle==0.1.*
pip install kivy
```

After these installations, your kivy should now be installed and ready to setup. Now let us set up our kivy project and start creating our applications. Follow the steps below to set up your kivy file:

1. Create a new Project on your PyCharm and save it as **Kivy_tut**.
2. Right click your file and click on new then create a new python file

and save it as **main**.

3. Click on `kivy_tut` and then go to file on the top left, then settings, then under the project you just created click on project interpreter and click in the plus sign on the right. The image below shows:



4. After clicking on the plus sign search for kivy and install the following packages.

- Kivy
- Kivy-examples
- Kivy Calendar
- Kivy-deps.angle
- Kivy-deps.glew
- Kivy-deps.gstreamer
- Kivy-deps.sdl2
- Kivymd

Well done! You have installed and set up Kivy on your project and are ready to start. Note that for every new project you will ever create, you have to do the same process of adding the interpreters above. Let us get started!

Section 4.2: First Kivy App

When creating an app, you will always have to start by creating a window, then filling it up with your desired outcome. We always create a first our app by starting out with 3 steps:

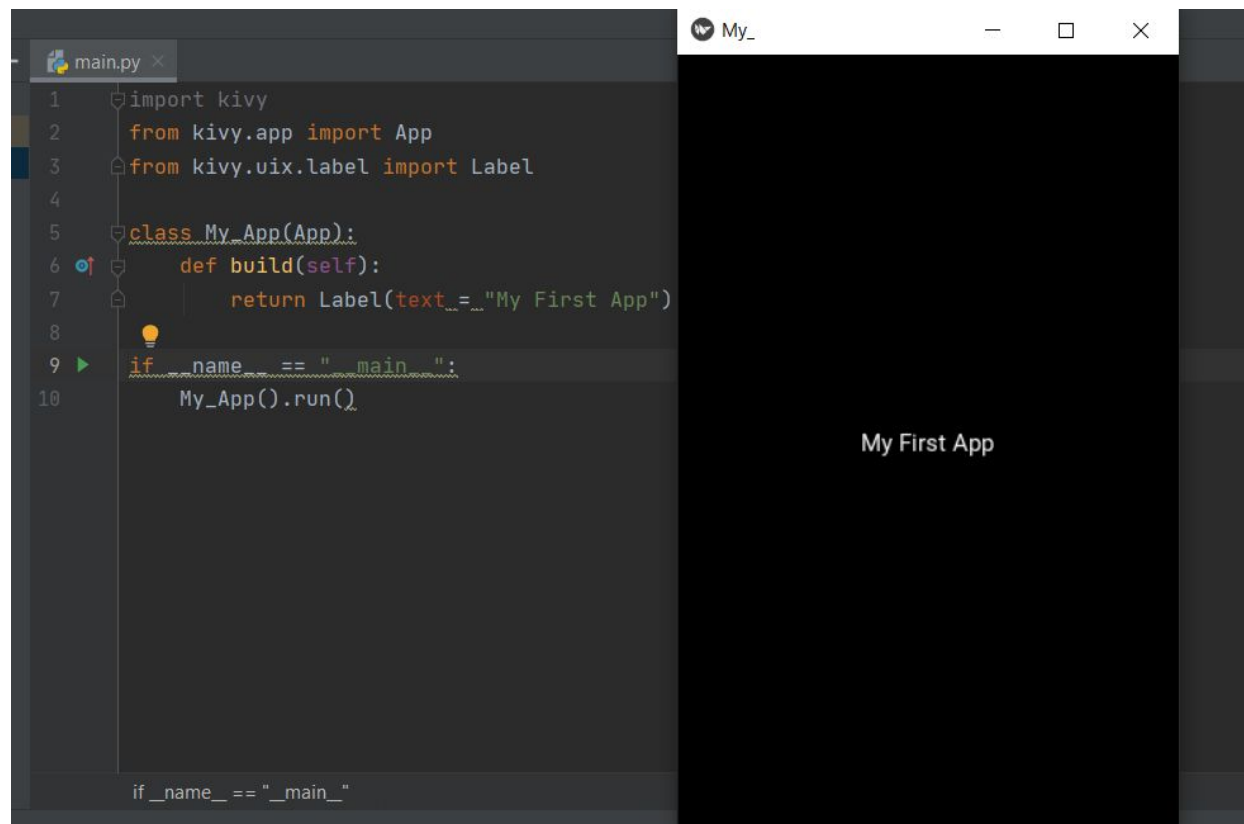
1. We import kivy
2. We import our App from kivy.app
3. We import a widget/s that we will be using in our app
4. We create our app class and subclass it with our imported App
5. We return our widget
6. We instantiate our app class with the **run()** method
7. We then fill up our app with desired outcomes.

Let us create our first simple app:

```
import kivy
from kivy.app import App
from kivy.uix.label import Label
```

```
class My_App(App):
    def build(self):
    return Label(text = "My First App") if __name__ == "__main__":
    My_App().run()
```

When you run the code, this should return you an app like the image below:



Well done! You have created your first application, let us now explain the code:

```
import kivy
from kivy.app import App from kivy.uix.label import Label
```

We first imported Kivy and then imported our App from kivy.app. This is essential for every kivy app you will ever create. Always start with those two.

We then imported our widget, which was our **Label** from kivy.uix, which is where we will always import all the widgets (Label, Button, Text input, Color, etc.) we will ever use. We will look at all the existing kivy widgets later in this chapter.

```
class My_App(App):
def build(self):
return Label(text = "My First App")
```

Here, we define the base class of our kivy app. Then we subclass our App inside the brackets of our class to inherit the App class we imported from kivy.app. We then initialize and return our widget using the **def build(self)** method and **return** method. Our widget is a label and we give that label a text ("My first App"). Our Widget could be anything

```
if __name__ == "__main__": My_App().run()
```

We now initialize our app with the **run()** method. I always recommend you always start by writing this code when creating any app and then customize and fill up your app. Know how to write this code from heart.

Before customizing our application, let us look at another example where our widget becomes our button:

```
import kivy
from kivy.app import App
from kivy.uix.button import Button
```

```
class My_App(App):
def build(self):
return Button(text = "My First App")
```

if __name__ == "__main__": We have replaced our label with a button and now you can click the button and your screen will turn blue to signify a click.

Let us go onto customizing our application.

Section 4.3 Customizing Our Application

Let us now customize our application by creating a sign up app screen that will let a user input his/her information and then submit it. Our sign up screen will not save any of this information, as we have not covered how to create a database for a sign up and login screen. This section is just to show you the interactions of widgets to creating our desired outcome.

```
import kivy
from kivy.app import App
from kivy.uix.gridlayout import GridLayout from kivy.uix.textinput
import TextInput from kivy.uix.label import Label
from kivy.uix.button import Button
```

```
class SignUp_Screen(GridLayout):
    def __init__(self,**kwargs):
        super(SignUp_Screen, self).__init__(**kwargs) self.cols = 2
```

```
self .add_widget(Label(text="First Name"))self.username =
    TextInput(multiline=False)
self.add_widget(self.username)
```

```
self .add_widget(Label(text="Last Name"))
self.username = TextInput(multiline=False)
self.add_widget(self.username)
```

```
self .add_widget(Label(text="Email"))
self.username = TextInput(multiline=False)
self.add_widget(self.username)
```

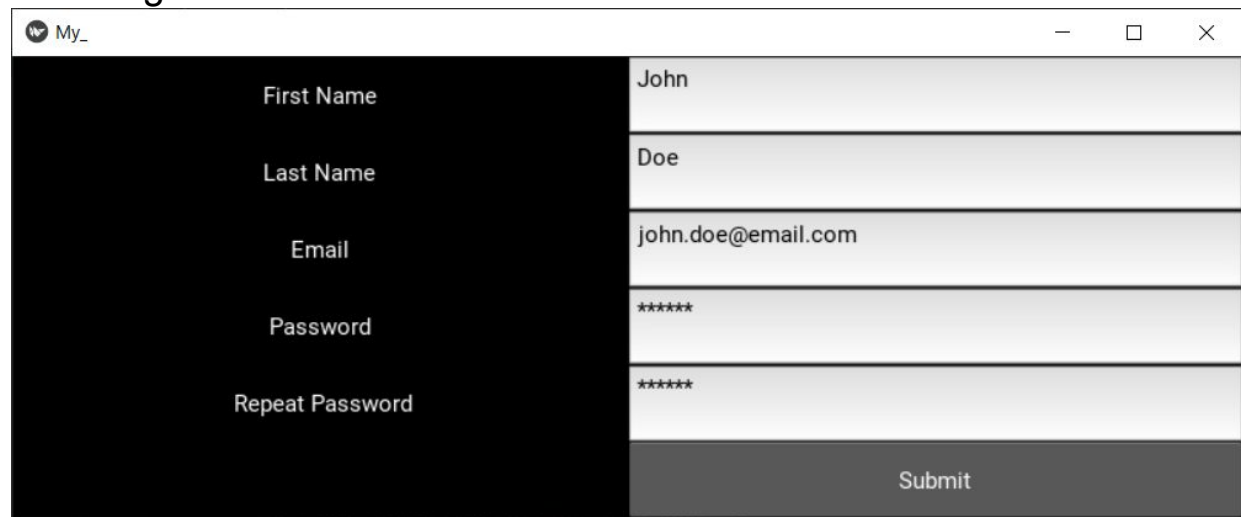
```
self .add_widget(Label(text="Password"))  
self.username = TextInput(multiline=False, password=True)  
self.add_widget(self.username)
```

```
self .add_widget(Label(text=" Repeat Password")) self.username =  
TextInput(multiline=False, password=True)  
self.add_widget(self.username)
```

```
self .add_widget(Label(text=""))  
self.submit = Button(text="Submit")  
self.add_widget(self.submit)
```

```
class My_App(App):  
def build(self):  
return SignUp_Screen()  
if __name__ == "__main__":
```

You have created your sign up screen; your code should return the following screen:



First Name	John
Last Name	Doe
Email	john.doe@email.com
Password	*****
Repeat Password	*****
Submit	

Now let us discuss the code:

```
import kivy  
from kivy.app import App  
from kivy.ui.gridlayout import GridLayout  
from kivy.ui.textinput import TextInput  
from kivy.ui.label import Label  
from kivy.ui.button import Button
```

We start by importing our kivy and our App from kivy.app. We then import all the Widgets from kivy.ui that we are going to need, which will be our grid layout (to build our app on), text input (to input the required information), label (to label our textbox), and button (to submit our application).

```
class SignUp_Screen(GridLayout):  
def __init__(self,**kwargs):  
super(SignUp_Screen, self).__init__(**kwargs) self.cols = 2
```

We then create our base class and subclass it with our grid layout because that is where our application will be built. We then define our an `__init__()` method so we can add and define the behavior of our widgets. It is good practice to always include the `**kwargs` (kivy widgets arguments) as we can sometimes need to use them internally. They are used to pass a variable number of arguments to a function. The `super()` method is to implement the functionality of the original class being overloaded.

We then tell the program how many columns our grid layout will consist of using the `self.cols` method. In this grid layout, we only needed two columns (label and text input).

```
self.add_widget(Label(text="First Name")) self.username =  
TextInput(multiline=False)self.add_widget(self.username)
```

```
self .add_widget(Label(text="Last Name")) self.username =  
TextInput(multiline=False) self.add_widget(self.username)
```

```
self.add_widget(Label(text="Email")) self.username =  
TextInput(multiline=False)self.add_widget(self.username)
```

```
self.add_widget(Label(text="Password"))
```

```
self.username = TextInput(multiline=False, password=  
True)self.add_widget(self.username)
```

```
self.add_widget(Label(text=" Repeat Password")) self.username =  
TextInput(multiline=False, password=True)self.add_widget(self.username)
```

```
self.add_widget(Label(text="")) self.submit =  
Button(text="Submit")self.add_widget(self.submit)
```

We now add all the widgets we need and label them. Our text-input boxes will not be multi touch so we equate multi touch to false and for the password and repeat password widgets; we equate the password to true so the password remains hidden. We then add a button to submit our details. Of course our information will not be submitted yet as we have not looked into creating databases. We will come back to this again later when we have looked at databases.

```
class My_App(App):  
def build(self):  
return SignUp_Screen() if __name__ == "__main__":  
My_App().run()
```

We know create our class app and subclass it with our App widget and then return the sign up screen class.

We then run our app and it will then return our sign up screen.

We should now have an understanding on how widgets work and how to build and import them. This should have also given you a deeper understanding on functions and classes that we learnt in the python section. A challenge that might arise while building an app could be not knowing which widget to import, this is where we talk about the importance of planning your app first as this can save you a lot of time when you build your app while following a structured plan. Let us look at the importance of planning in the following section.

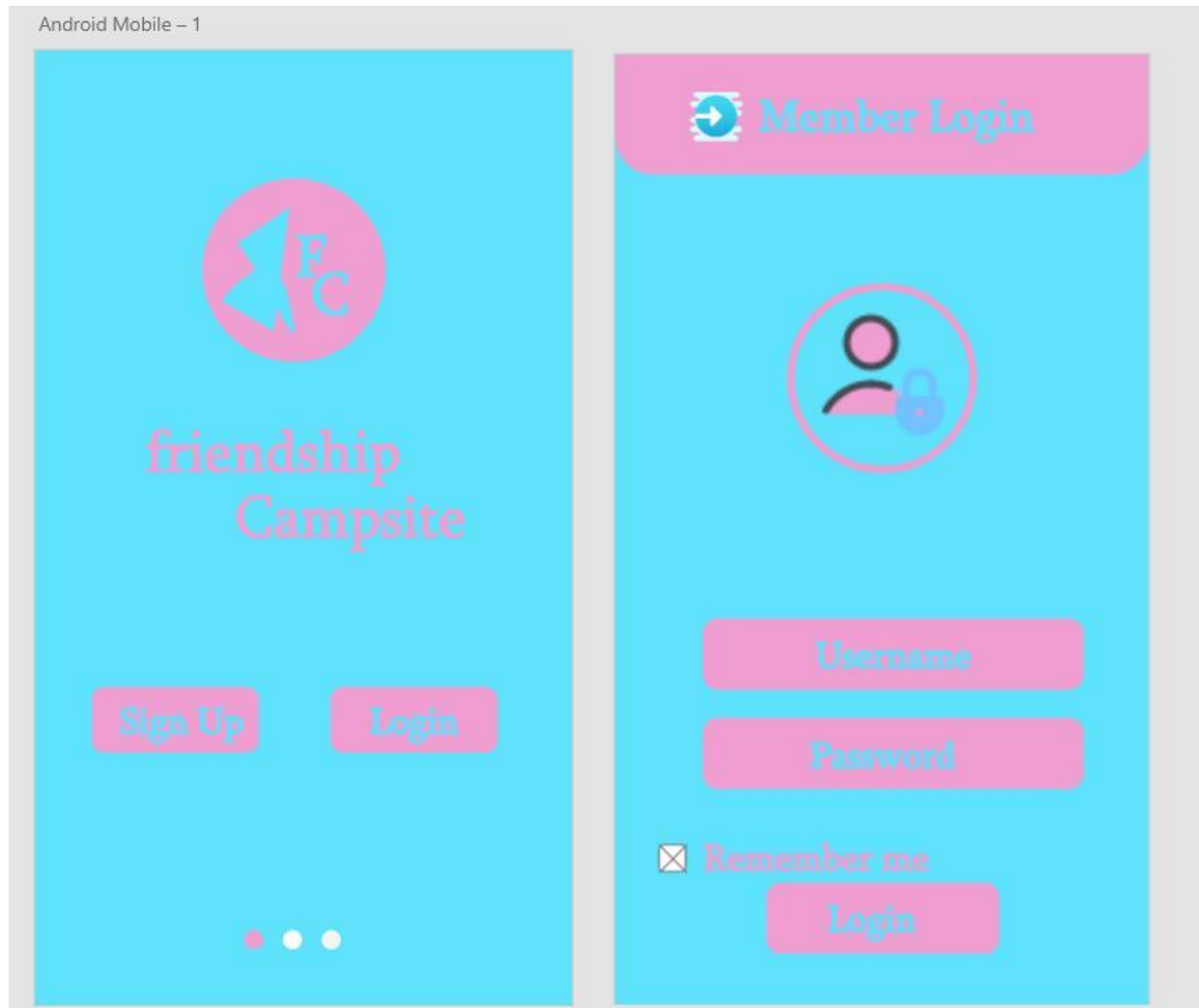
Section 4.4: Planning Your App

“Planning is the process of thinking about the activities required to achieve a desired goal. It is the first and foremost activity to achieve desired results. It involves the creation and maintenance of a plan, such as psychological aspects that require conceptual skills.”

-Wikipedia

There are some software's that you can use to plan your app like Adobe XD that allow you to design your app's UI before you go onto creating it. You do not need to know how to code in order you use

Adobe XD, it is a simple drag and drop software. Below is an example of an app UI that I have designed with Adobe XD:



Although adobe XD is a good software to create your app's UI, I still recommend you also design your app with pen and paper for proper planning so you can also write the structure of your app down next to every plan and the details of what you want every widget to do. That way, you will have a good plan and build you app successfully.

You can download adobe XD at
<https://creativecloud.adobe.com/apps/download/xd>

Section 4.5: Concept Discussion of chapter 4

Under the UIX, we have widgets and layouts. **Widgets** are the base building blocks of the graphic user interfaces in our apps and **layouts** are what we use to arrange widgets.

We have two types of widget events:

1. **Widget-defined events**: events that will be fired when a button is pressed or released 2. **Property events**: events fired when your widget changes their size or position. We have six types of layouts:

1. **Floatlayout**: Allows placing children with arbitrary locations and size
2. **Boxlayout**: Arranges widgets in an adjacent manner
3. **Gridlayout**: Arranges widgets in a grid.
4. **Stacklayout**: Arranges widgets adjacent to one another
5. **Relativelayout**: Behaves just like FloatLayout, except children positions are relative to layout position
6. **Anchorlayout**: layout only caring about children positions.

UI is a user interface (the dashboard where everything on the app is found.)

Section 4.6: Property events in Kivy

We use properties to define events and abide to them in kivy. We have different kinds of events, below is a list:

- StringProperty
- NumericProperty
- BoundedNumericProperty
- ObjectProperty
- DictProperty
- ListProperty
- OptionProperty
- AliasProperty
- BooleanProperty
- ReferenceListProperty

We will look at these properties in detail as we use them in our code as we go on. We declare properties at the class level, which will then instantiate when your object is created.

Button Clicks

Let us now go back to our sign up screen and add a widget-defined event that will print out the statement “Thank you for submitting” every time our submit button is clicked:

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

```
from kivy.uix.button import Button
```

```
class SignUp_Screen(GridLayout):
    def __init__(self,**kwargs):
        super(SignUp_Screen, self).__init__(**kwargs)
        self.cols = 2
```

```
        self .add_widget(Label(text="First Name"))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
```

```
        self .add_widget(Label(text="Last Name"))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
```

```
        self .add_widget(Label(text="Email"))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
```

```
        self .add_widget(Label(text="Password"))
        self.username = TextInput(multiline=False, password= True)
        self.add_widget(self.username)
```

```
self .add_widget(Label(text=" Repeat Password")) self.username =  
TextInput(multiline=False, password=True)  
self.add_widget(self.username)
```

```
self .add_widget(Label(text=""))  
self.submit = Button(text="Submit")  
self.add_widget(self.submit)  
button_one= self.submit
```

```
def callback(self,instance):  
print("Thank you for submitting")  
button_one.bind(state=callback)
```

```
class My_App(App):  
def build(self):  
return SignUp_Screen() if __name__ == "__main__":
```

```
My_App().run()
```

This code will now return a statement that says “Thank you for submitting” when you run the code. Let us explain what we added:

```
self .add_widget(Label(text="")) self.submit =  
Button(text="Submit") self.add_widget(self.submit)  
button_one= self.submit
```

We started by creating a new variable (button_one) and assigning it to our existing button (self.button).

```
def callback(self,instance):  
print("Thank you for submitting")  
button_one.bind(state=callback)
```

We then define a new function and add an instance as a parameter. We bind our button_one and assign our function to the state, which is an OptionProperty that defaults to “normal”.

You can customize your widgets by adding a color to them, a font_size, vertical alignment (valign), horizontal alignment (haling), a size (size_hint), position (pos_hint), etc. we will learn how to apply these widgets when we start creating our applications.

Let us look at an example, by customizing our button from the sign up screen by changing its background color:

```
self.add_widget(Label(text=""))  
self.submit = Button(text="Submit", background_color =  
(1.0,0.0,3.0,1.0)) self.add_widget(self.submit)  
button_one= self.submit
```

Note that our background color should be in rgba. If you do not know the colors in rgba, simply translate your color from English to rgba using google.

You can also put add background images, but you would first need to add the image to your file, the import Images from kivy.uix.image. we will only cover this when we create a project in the upcoming chapters.

You can also add a background color to your whole app using the method we used in the above code. We will also cover this in detail in our projects in the upcoming chapters.

Section 4.7: Canvas

The Canvas is the root object used for drawing by a widget. To use Canvas, you have to import, e.g.

```
from kivy.graphics import Rectangle, Color
```

We normally use canvas to add an instruction to widget, we will not go into detail yet as we first have to have an understanding of the next section.

Section 4.8: The KV language

The kivy language (.kv), allows you to create your widget tree in a declarative way and to bind widget properties to each other or to callbacks in a natural manner. The kivy language may be useful to organize and separate application logic from the presentation of it, think of it as HTML and CSS.

We use the kivy language by writing our widgets on our main python file, but kivy its instruction on a separate file. To open a kv file, right click on your project file (which we had initially named kivy_tut) and create a new file (**not python file, just file**) and name it **main.kv**.

Let us now look at an example from the first example we did in this chapter:

We already have our python file (main.py). Now create a kivy file and name it **My_App.kv**

```
import kivy
from kivy.app import App
from kivy.uix.label import Label

class My_App(App):
    def build(self):
        return Label()

if __name__ == "__main__":
    My_App().run()

<Label>
text : "My First App"
```

A



A is our python file (main.py) and B is our kivy file (My_App.kv). Note that our kivy file is name after the class My_App since that is where it will be instructed.

Do not forget to tab in or space four time when going into the new line under <label> as our text is directed to the label.

You will get more understanding when we get to do a hands-on approach by building apps.

Section 4.9: Touch methods

We use touch events to go certain functions in our programs like moving objects, printing out statements after clicks, calling actions, changing between pages, etc. We have three different types of touch events:

On_touch_down: An event fired when a click or touch down event has been initiated. **On_touch_up:** An event fired after a click or touch event is released, this means it will only be done when you let go after a click or press.

On_touch_move: An event fired when a touch even is moved on changes location

Let us now look at a more practical approach as an example:

```
import kivy
from kivy.app import App
from kivy.uix.widget import Widget from kivy.core.window import Window
```

```
class App_Screen(Widget):
def on_touch_down(self, touch):
if "button" in touch.profile:

self.ids.button_one.text = touch.button
```

Main.py



```
class My_App(App):
Window.size = (300,500)
def build(self):

return App_Screen()
if __name__=="__main__": My_App().run()
<App_Screen>:
```

Button: `My_App.kv` id: `button_one`

```
text: 'Click me'
```

We start by importing our app class widget and window from kivy. We basically use `window.size` to try make our application look the size of a mobile phone so we see what it would look like of a mobile phone, but when packaging your app; always be sure to remove that otherwise your apk will have trouble when you import it to a phone. We created the `app_screen` class and sub classed it with the widget and this is where our app functions are called. We then went into the kivy file and created a button. Note that we call our class using the html tag (`<App_class>`:) . it is necessary to put an id in buttons when we create them so that we can later use the id to call certain functions of it.

def `on_touch_down`(self, touch):

Here, we declare the dispatched **on_touch_down** method. note the parameter `touch` in the declaration, and this parameter is necessary to call the event using the input

if “**button**” in `touch.profile`:

This is basically a verification line. We use it to be sure that the input device used in the platform, where we are running our code, supports the button profile. Without this line, there are possibilities of the app crushing.

```
self.ids.button_one.text = touch.button
```

This is where we call the button profile and let the button change its text from “click me” to either left or right depending on whether your left-click or right-click it.

class `My_App`(App):

```
Window.size = (300,500) def build(self):
```

```

return App_Screen()
if __name__ == "__main__": My_App().run()

```

The rest of the code we have previously discussed it. I would like to remind you never forget to name your kivy file the same of your app class, this canes being "My_App.kv". Without this being correct, your kivy file will never be called into the python file.

If you would like the button to show you the position it was clicked at instead of changing text, you can simply replace "touch.button" with "str(touch.pos)" :

```

self.ids.button_one.text = str(touch.pos)

```

Section 4.10: Multi-touches

An application can have a multi-touch like a double or triple press of objects or buttons. We usually see this in games. Multi-tapping dictation can be very useful in app development in case you want to add certain functionalities to your application. Let us look at an example of a double tap using the on_touch_down method:

```

import kivy
from kivy.app import App
from kivy.core.window import Window from kivy.uix.widget import
Widget

```

```

class App_Screen(Widget):
    def on_touch_down(self, touch):
        if touch.is_double_tap:
            self.ids.label_one.text = "Double tapped - The interval is: " +
            str(touch.double_tap_time)

```

```

class My_App(App):
    Window.size =(300,500) def build(self):

```

```

return App_Screen()
if __name__ == "__main__": My_App().run()
<App_Screen>:

```

Button:

id: button_one pos: 0,0

text: "Click Me"

Label:

id: label_one pos: 150, 150 text: "

Our program will now return an app screen with a button labelled "Click Me", when clicked the button will return a statement that says, "Double tapped" with the interval. Let us look more into the code:

We have already explained the dispatching of the on_touch_down, so we will start from the next line from there:

```
if touch.is_double_tap:
```

```
self.ids.label_one.text = "Double tapped - The interval is: " +  
str(touch.double_tap_time)
```

Here we use a conditional statement to tell the program that when it dictates a double touch, it should bring up the statement "Double tapped" and the time the double touch takes.

Section 4.11: Events

An event is something that happens and GUI frameworks usually provide ways to handle these events, in this case our GUI framework being kivy. We will cover three main types of events: Clock-related events: Events that are ran by time and time plays an important role.

Property-events: Events that are fired when property events are changed such as size or position events

Widget- related events: Events related to a widgets' functionality.

We had already discussed property and widget, so we will not discuss them in this section.

Scheduling a one-time event:

A one-time event is set to be performed just once after a certain time interval has passes. Let us take a look at an example:


```

import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.clock import Clock

class App_Screen(Widget):
def my_callback(self, dt):
self.ids.label_one.text = "callback has been called" + str(dt)
def on_touch_down(self, touch):
Clock.schedule_once(self.my_callback, 1)
class My_App(App):
def build(self):
return App_Screen()
if __name__ == "__main__": My_App().run()
<App_Screen>:

```

```

Label:
id: label_one pos: 150,150 text: ""

```

This will return a blank screen, but after a it has been clicked an event will be called after one second; this is because we set our callback to one second. Let us closely discuss the code

```

from kivy.clock import Clock

```

We always need to import the clock object in order to schedule an event.

```

def my_callback(self, dt):

```

We define the function that will be called when the event is fired.

```

self.ids.label_one.text = "callback has been called" + str(dt)

```

This is the string we want called after the interval of the scheduled time. Clock.schedule_once(self.my_callback, 1)

here we schedule a one-time event, and this event is to call the function method my_callback. It will occur after a second has passed, and then Clock.schedule_once() is called.

Scheduling a repetitive event:

A repetitive even is one that will continuously be called after a certain number of seconds. Below is an example:

```

import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.clock import Clock

class App_Screen(Widget):
    def my_callback(self, dt):
        self.ids.label_one.text = "callback has been called" + str(dt)

    def on_touch_down(self, touch):
        Clock.schedule_interval(self.my_callback, 1 / 30.) if
        touch.is_double_tap:

        self.ids.label_one.text = "
        Clock.unschedule(self.my_callback)
class My_App(App): def build(self):
return App_Screen() if __name__ == '__main__': My_App().run()
<App_Screen>:

```

Label:
id: label_one pos: 150,150 text: ""

This program will return an empty screen, but when pressed will bring a continuous call of events and to stop it you need to double tap.

Looking at the code more closely (only at what has not been covered):

```
Clock.schedule_interval(self.my_callback, 1 / 30.)
```

Here, we schedule the repetitive event where the callback function is called and the event will be performed 30 times per second. We will dive deeper into this when we build the pong game.

if touch.is_double_tap: here we select the double taps made in our app.

Section 4.12:Advanced Graphics

We had already looked at graphics when we spoke about canvas, rectangles,multi-touching etc. now we look into more graphics that can be used to better our mobile applications.

Using Carousel

The carousel is a widget that allows you to swipe between slides. It is an advanced way to present pictures

An example is below:

```
from kivy.app import App
from kivy.uix.carousel import Carousel
from kivy.uix.image import AsyncImage
from kivy.core.window import Window

class My_App(App):
    Window.size = (300,500)
    def build(self):

        carousel = Carousel(direction='right')
        for i in range(10):
            src = "http://placeholder.it/480x270.png&text=slide-%d&.png" % i
            image = AsyncImage(source=src, allow_stretch=True)
            carousel.add_widget(image)
        return carousel

if __name__=="__main__": My_App().run()
```

This program will return a window for us that will have image placeholders that let us slide within them both left and right. You can add you won images by adding some pictured to your file and using them instead.

Asynchronous data Loader:

Asynchronous data loaders are used to load an image and use it, even if data are not yet available.

Example:

```
from kivy.loader import Loader image =
Loader.image('myimage.png')
```

You can also load an image from a url:

```
image = Loader.image('http://mysite.com/test.png')
```

You can load an image using the Loader. A ProxyImage is returned with a loading image. An example is below:

```
from kivy.app import App
from kivy.ui.image import Image from kivy.loader import Loader
from kivy.core.window import Window

class My_App(App):
    Window.size = (300,500)
    def _image_loaded(self, proxyImage):

if proxyImage.image.texture:
    self.image.texture = proxyImage.image.texture

    def build(self):
        proxyImage = Loader.image("mk.jpg")
        proxyImage.bind(on_load=self._image_loaded) self.image = Image()
        return self.image

if __name__=="__main__": My_App().run()
```

This will return your screen with your image. Make sure you have an image in your folder that you call out otherwise; it will just return an empty white box.

Chapter 5: Creating Mobile Apps

By now, we have learnt many building blocks to be able to build a full mobile app and package it then deploy it to play-store. In this chapter, we will build a few mobile applications of different functionalities in order to gain the experience of mobile app development and use all the building blocks of what we have learnt to make any app of our desire. Before that, I would like to go through some extended concepts where we discuss widget control and manipulations. This will better your understanding in the mobile apps we will build

Section 5.1: Widget Manipulations

We will look at these widget manipulations through hands-on application more than theory. To start off, create your **main.py** file and a kivy file and call it **My_App.kv**. if you already have this files, just clear everything inside and use them.

Button Behaviors:

Let us look at the possibilities we can do with buttons. Let us start by creating our new window:

```
import kivy
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.core.window import Window
from kivy.uix.label import Label

class Main_Screen(Widget):
```

pass **Main.py**

class My_App(App):



Window.size = (300, 500)

def build(self):

return Main_Screen()

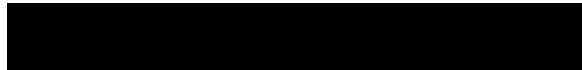
if __name__=="__main__": My_App().run()

<Main_Screen>:

canvas:

Rectangle: My_App.kv

pos: self.pos



size:self.size

We have created our app's screen that is empty. Note that the kivy file is names after the call that inherited the app class (My_App(App)), this should always be the case. We also returned our main_screen class because that is where we will be building our app. We used the **pass** method to tell the program to skip that for now is it would be used later on. The **window.size** tells the computer the size of your screen we want returned. We use this method just on the program to see how our app would look like on a mobile phone, but when packaging our app we do not include the method. We always need to import the window function from kivy.core

Let us now add our button:

```
import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.core.window import
Window
```

```
class My_Button(Widget): pass
class Main_Screen(Widget): pass
```

```
class My_App(App):
Window.size = (300, 500) def build(self):
```

```
return Main_Screen()
if __name__=="__main__": My_App().run()
<Main_Screen>: canvas:
```

```
Rectangle:
pos: self.pos size:self.size
```

```
My_Button: size: self.size pos:self.pos
```

```
<My_Button>:
```

```
Button:
size: 220,60 pos: 40,20
text: "Click Me" We did this by first creating a class for our button and
then customizing it in the kivy file. Note that although we call our
button and customize it in the kivy file, we still call in again under the
<Main_Screen>: in the kivy file. This is because we return the
main_screen on our app class in the python file, so we need to
include everything in the main_screen in kivy too. We used pos:
self.pos and size:self.size in the kivy file to tell the program we are
taking the position and size we had already set.
```

Printing a statement when button clicked:

Let us now print a statement when the button is clicked: To do this you simply need to add an on_press = "sentence" to your kivy.

```
<My_Button>:
```

```
Button:
size: 220,60
pos: 40,20
text: "Click Me"
color: [0, 84, 80, 19]
on_press: print("I've been pressed")
```

We have also added a color to our buttons' label. No changes are made in the python file

Box Layout & Text Input:

Let us further customize our app by adding a text input box that will let us type anything in it and change our buttons' label to "submit": For this we would need to now import boxlayout and textinput from kivy:

```
import kivy
from kivy.app import App
from kivy.uix.widget import Widget from kivy.core.window import
Window from kivy.uix.textinput import TextInput from
kivy.uix.boxlayout import BoxLayout

class Main_Screen(Widget):
pass
class My_Button(Widget):
pass

class My_Text(BoxLayout): pass

class My_App(App):
    Window.size = (300, 500) def build(self):

return Main_Screen() if __name__=="__main__": My_App().run()
<Main_Screen>: canvas:
```

```
Rectangle:
pos: self.pos size:self.size
```



```
My_Button:  
id: button_one size: self.size pos:self.pos
```

```
My_Text:  
pos:40,220  
size:220,60
```

```
<My_Button>:
```

```
Button:  
size: 220,60  
pos: 40,160  
text: "Click Me"  
color: [0, 84, 80, 19]  
on_press: print("Submitted")
```

```
<My_Text>:
```

```
BoxLayout:  
pos:150,220 size:220,20 TextInput:  
  
size:220,20
```

We added a new class for our input text and let it contain the pass method. We then went into our kivy folder and customized how our box layout would be sized, positioned and sized the text input. Note that we then added it to the main_screen since we return mainscreen. The positions will always be different if you use numbers as device sizes differ, that is why when we start building our mobile apps in the next section you will notice we choose to use the root.sizes.

Let us now build a few sample apps and we will cover more through their developments

Section 5.2: A Simple Pong Game

Let us start by building a simple pong game. We will build it step by step and display the full source code at the end of the tutorial.

First, let us plan our app:

1. We want a screen with a division line, player one and player two sides
2. We want to add our paddles, one on each side
3. We want to add our ball
4. We want to move our paddles
5. We want to move our ball
6. We want to let the ball bound up and down
7. We want to let the ball bounce of paddles

Adding the screen:

```
import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.core.window import Window
```

```
class Game_Screen(Widget): pass
```

```
class My_App(App):
    Window.size = (300,500) def build(self):
```

```
    return Game_Screen()
if __name__=="__main__": My_App().run(
    <Game_Screen>:
    canvas:
```

```
    Rectangle:
    pos:self.center_x-4,0 size:8,self.height
```

We have now added our screen and the division line in-between.
Note that for the division line we said `self.center_x - 4`, this is because we gave it a size width of eight so we subtract half to make it centered.

Adding the paddles:

```

import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.core.window import
Window

```

```

class Game_Screen(Widget): pass
class Game_Paddles(Widget): pass

```

```

class My_App(App):
Window.size = (300,500) def build(self):

```

```

return Game_Screen()
if __name__=="__main__": My_App().run()
<Game_Screen>:
canvas:

```

```

Rectangle:
pos:self.center_x-4,0 size:8,self.height

```

```

Game_Paddles:
id: player_left
size:20,200
x:root.x
center_y:root.center_y

```

```

Game_Paddles:
id: player_right
size: 20,200
x: root.width - self.width center_y: root.center_y

```

```

<Game_Paddles>: canvas:

```

```

Rectangle:
size: self.size pos:self.pos

```

We now added a new class of game paddles and let it pass. We then went into the kivy file and created our game paddles. As you can see, we chose our position in terms of x and y. this is because we want to make our app suitable in any device by taking the device's root size and not a manual put size as that can cause problems later.

We also put ids in our paddles; this is to make it easy for us to later call them in functions. Ids are like variables, except they contain names on objects or widgets we put.

Adding the ball:

```
import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.core.window import
Window
```

```
class Game_Screen(Widget): pass
class Game_Paddles(Widget): pass
class Game_Ball(Widget): pass
```

```
class My_App(App):
Window.size = (300,500) def build(self):
```

```
return Game_Screen()
if __name__=="__main__":
<Game_Screen>:
canvas:
```

```
Rectangle:
pos:self.center_x-4,0 size:8,self.height
```

```
Game_Paddles:
id: player_left
size:20,200
x:root.x
center_y:root.center_y
```

```
Game_Paddles:
id: player_right
size: 20,200
x: root.width - self.width center_y: root.center_y
```

```
Game_Ball:  
id: game_ball  
center:self.parent.center
```

```
<Game_Paddles>: canvas:
```

```
Rectangle:  
size: self.size pos:self.pos
```

```
<Game_Ball>:  
size:40,40  
canvas:
```

```
Ellipse:  
pos: self.pos size: self.size
```

Here, we also create a class for our game ball and just pass it for now. We then go into our kivy file. First, note that for our ball we used an ellipse and now a rectangle for our canvas. This is because we want our ball to be a circle. Also note that we put the balls position at the parent center, this is because if you just say self.center it will take the not have any effect as it would only use if widget for the center. Alternatively you can use root.center instead of self.parent.center

Now we have our pong game set up, let us start moving objects around.

Moving the paddles:

```
<Game_Screen>:  
ball: game_ball  
player_one:player_left player_two:player_right canvas:
```

```
Rectangle:  
pos:self.center_x-4,0 size:8,self.height
```

we start by assigning the ids to variables we will use in the game.

```

import kivy
from kivy.app import App
from kivy.ui.widget import Widget from kivy.core.window import
Window from kivy.properties import ObjectProperty

class Game_Screen(Widget):
    player_one = ObjectProperty(None) player_two =
    ObjectProperty(None)

    def on_touch_move(self, touch): if touch.x < self.width/3:
    self.player_one.center_y = touch.y if touch.x >self.width- self.width/3:
    self.player_two.center_y = touch.y
    class Game_Paddles(Widget): pass
    class Game_Ball(Widget): pass

class My_App(App):
    Window.size = (300,500) def build(self):

    return Game_Screen()
if __name__=="__main__":

```

We start by import object property from kivy.properties. This is because we want our paddles to be objects In order to move them. This is why both players were equated to object properties in the game screen class.

We then use a conditional statement for movement. For player one, we tell the program that if the player touches anywhere in less than 1/3 of the left of the program, he can move the player one paddle and the opposite does the same for player two. We use the on_touch_down method for any object that we want to move by touching.

```

<Game_Screen>:
ball: game_ball
player_one:player_left player_two:player_right canvas:

```

```

Rectangle:
pos:self.center_x-4,0 size:8,self.height

```

Game_Paddles:
id: player_left
size:20,200
x:root.x
center_y:root.center_y

Game_Paddles:
id: player_right
size: 20,200
x: root.width - self.width center_y: root.center_y

Game_Ball:
id: game_ball
center:self.parent.center

<Game_Paddles>: canvas:

Rectangle:
size: self.size pos:self.pos

<Game_Ball>:
size:40,40
canvas:

Ellipse:
pos: self.pos size: self.size

There are no changes in the kivy file.

Moving the ball:

```
import kivy
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.core.window import Window
from kivy.properties import ObjectProperty, NumericProperty, \
```

```

ReferenceListProperty
from kivy.vector import Vector
from kivy.clock import Clock
from random import randint

class Game_Screen(Widget):
    ball = ObjectProperty(None) player_one = ObjectProperty(None)
    player_two = ObjectProperty(None)

    def start_ball(self):
        self.ball.center = self.center
        self.ball.velocity = Vector(4,0).rotate(randint(0,360))

    def update(self,dt): self.ball.move()
    def on_touch_move(self, touch): if touch.x < self.width/3:
        self.player_one.center_y = touch.y if touch.x >self.width- self.width/3:
        self.player_two.center_y = touch.y
    class Game_Paddles(Widget): pass

class Game_Ball(Widget):
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)
    velocity = ReferenceListProperty(velocity_x,velocity_y)

    def move(self):
        self.pos = Vector(*self.velocity) + self.pos

class My_App(App):
    Window.size = (300,500)
    def build(self):

        game = Game_Screen()
        game.start_ball()
        Clock.schedule_interval(game.update, 1.0/58.0) return game

if __name__=="__main__":

    <Game_Screen>:
    ball: game_ball

```


player_one:player_left player_two:player_right canvas:

Rectangle:

pos:self.center_x-4,0 size:8,self.height

Game_Paddles:

id: player_left

size:20,200

x:root.x

center_y:root.center_y

Game_Paddles:

id: player_right

size: 20,200

x: root.width - self.width center_y: root.center_y

Game_Ball:

id: game_ball

center: self.parent.center

<Game_Paddles>: canvas:

Rectangle:

size: self.size pos:self.pos

<Game_Ball>:

size:40,40

canvas:

Ellipse:

pos: self.pos size: self.size

The ball now moves in a random direction when you run the program. It does not have any boundaries yet, so it keeps going even when it is out of the screen. Let us explain the code. **class**

Game_Screen(Widget):

ball = ObjectProperty(**None**) player_one = ObjectProperty(**None**)

player_two = ObjectProperty(**None**)

```

def start_ball(self):
self.ball.center = self.center
self.ball.velocity = Vector(4,0).rotate(randint(0,360))

def update(self,dt): self.ball.move()
def on_touch_move(self, touch): if touch.x < self.width/3:
self.player_one.center_y = touch.y if touch.x >self.width- self.width/3:
self.player_two.center_y = touch.y

```

We start by assigning our ball as an object property; this is to be able to move it. We define a method to start the ball and we tell the program that the velocity of our ball will be in vector format(x and y-axis). We then use the rotate function so our ball will go on rotation and we use the random method so the ball start in a random direction; this is to bring fairness to the game.

We then create an update method and put dt (time) as an argument. We use this update method in the clock method so the program is updated a certain number of time. Let us look at the code below:

```

class My_App(App):
Window.size = (300,500)
def build(self):

game = Game_Screen()
game.start_ball()
Clock.schedule_interval(game.update, 1.0/58.0) return game

```

The clock object allows us to be able to schedule a function call in the frame; once or repeatedly at specified intervals. You can get the time elapses between the scheduling and the calling of the callback via the dt argument.

```

Clock.schedule_interval(game.update, 1.0/58.0)

```

This means we want the ball to appear 58 times in 1 second. This is so the ball will always seem to be moving, although its frames of the ball in 58 different positions.

```

class Game_Ball(Widget):

```

```
velocity_x = NumericProperty( 0)
velocity_y = NumericProperty(0)
velocity = ReferenceListProperty(velocity_x,velocity_y)
```

```
def move(self):
self.pos = Vector(*self.velocity) + self.pos
```

Under the game ball class, we assign our velocity_x and velocity_y as numeric proper and initially put them as zero; this is so we can be able to change the velocity of our ball. We then create a velocity variable and assign it to a reference list property and reference the x and y velocity; this is so our ball can be able to move in the same velocity in any random direction.

There are no changes in the kivy file.

Bouncing the ball up or down:

```
def update(self,dt): self.ball.move()
if (self.ball.y < 0) or (self.ball.top > self.height): self.ball.velocity_y *=
1
```

We use a conditional statement to tell the program that if the ball's y is less than zero or is greater than the height of the screen, it will go in the reverse direction. This is all we need for the ball to start bouncing up and down.

Bouncing the ball off the paddles:

```
def update(self,dt): self.ball.move()
if (self.ball.y < 0) or (self.ball.top > self.height): self.ball.velocity_y *=
1
if (self.ball.x <0) or (self.ball.right>self.width): self.ball.velocity_x *= 1
We start by making the ball also bounce left and right, regardless of
the paddles. Now, the ball has been bounded all around. Let us now
customize this further to bounce off the paddles. class
Game_Paddles(Widget):
```

```

def ball_bounces(self,ball):
if self.collide_widget(ball):
vx,vy = ball.velocity
offset = (ball.center_y - self.center_y)/(self.height/2) bounced =
Vector(-1* vx,vy)
vel = bounced*1.0
ball.velocity = vel.x, vel.y + offset

```

We start by defining a method under our game paddles class and we use the collide method under our conditional statement. This will not make the ball bounce yet because we haven't included this in our update method. Let us now do so:

```

def update(self,dt): self.ball.move()
if (self.ball.y < 0) or (self.ball.top > self.height): self.ball.velocity_y *=
-1
if (self.ball.x <0) or (self.ball.right>self.width): self.ball.velocity_x *= 1
self.player_one.ball_bounces(self.ball)
self.player_two.ball_bounces(self.ball)

```

We have now told the program under the update method that the ball will bounce. Our ball should now be able to bounce off the paddles. Although our ball now bounces off the paddles, it still is not interesting because the ball still bounces off the edge when paddles do not catch it; this prevents players scoring goals.

Exercise 5.2:

- a. Add labels to both sides of the division line that shows the scores of both players and each score of a player should increase when a player scores. A player should be able to score when the opponent fails to bounce the ball off the paddle. After a player score, the ball game to reset and the ball begins in the middle again.
- b. Let the game be a best of ten. The first player to get to ten, the program must print out which player wins.
- c. Add a background image to make you app look nicer.

Section 5.3: Mobile Calculator

Let us now create a mobile calculator that we will be able to use on our mobile phones. We will start with the full code and then explain it. We will create our calculator using the python and kv language so we understand how the kv language and file works.

Let us start by planning our application:

1. We want our UI to have the buttons 0 to 9, the equal sign(=), the point sign (.), the division sign(/), the multiplication sign(*), the addition sign(+) and the clear sign(AC).
2. We want the user to be able to enter the values and the results to be returned
3. We want to make sure it works ☐ ☐
4. We create our python file (calc.py)
5. We create our text file (calc.kv)

Calculator: python file

```
import kivy
from kivy.app import App
from kivy.uix.button import Button from kivy.uix.boxlayout import
BoxLayout from kivy.core.window import Window

class CalcWindow(BoxLayout): def __init__(self,**kwargs):
super().__init__(**kwargs) Window.size = (300,500)

numb = [7,8,9,"+",4,5,6,"-",1,2,3,"*", ".",0,"(",")","/"] self.numbers =
self.ids.numbers
numbc = ["C", "CA"]
self.numbers = self.ids.numbers for num in numb:

btn = Button(text=str(num),font_size = "30px", background_color =
(1.0,2.0,3.0,0.6)) btn.bind(on_release=self.cancel_numbs)
self.numbers.add_widget(btn)
```

```

for num in numbc:
    btn = Button(text=str(num),font_size = "30px", background_color =
    (1.0,0.0,0.0,1.0)) btn.bind(on_release=self.cancel_numbs)
    self.numbers.add_widget(btn)

eq = Button(text="=", font_size = "30px", background_color =
(1.0,0.0,0.0,1.0)) eq.bind(on_release = self.eval_numb)
self.numbers.add_widget(eq)

def cancel_numbs(self,instance): inputt = self.ids.inputt
inputt.text += instance.text

if instance.text == "CA": inputt.text = ""
elif instance.text == "C": inputt.text = inputt.text[:2]

def eval_numb(self,text): inputt = self.ids.inputt exp =inputt.text
eva = eval(exp)
inputt.text = str(eva)

class CalcApp(App):
def build(self):
return CalcWindow() if __name__=="__main__": CalcApp().run()

```

Calculator: Kivy File

```

<CalcWindow>:
orientation:"vertical" canvas.before:

```

```

Rectangle:
pos: self.pos size: self.size source: "pic.jpg"

```

```

BoxLayout:
id: textinput
size_hint_y: .1
TextInput:

```

```

id: inputt
size_hint_y:

```

```
padding: 10
font_size: 17
background_color: (1.0,1.0,1.0,1.0) focus: True
```

```
BoxLayout:
id: main_btn size_hint_y: .8
```

```
GridLayout: id: numbers cols:4
spacing: 7
```

```
BoxLayout:
id: title
size_hint_y: .1
Label:
```

```
text: "Musa's Calculator" font_size: 30
bold : True
```

We created two files and wrote the codes. On the python file, we wrote the structure of the code and on the kivy file, we wrote the design on the code (just like html and css). Now let us look the code in detail and explain it.

```
import kivy
from kivy.app import App
from kivy.uix.button import Button from kivy.uix.boxlayout import
BoxLayout from kivy.core.window import Window
```

We start by importing everything we will need. The window was imported from kivy.core in order to give our window a specific size.

```
class CalcWindow(BoxLayout): def __init__(self,**kwargs):
super().__init__(**kwargs) Window.size = (300,500)
```

```
numb = [7,8,9,"+",4,5,6,"-",1,2,3,"*", ".",0,"(",")","/"] self.numbers =
self.ids.numbers
numbc = ["C", "CA"]
self.numbers = self.ids.numbers
```

We then create a class of our calculator window and subclass it with the box layout. Inside it we define our function and we create our window size to a height of 500 and width of 300 (usually a good size for mobile apps). We then create the list of our buttons and assign them to a variable "numb". We create our other two buttons (C & CA) in a different variable, because they will have different functionalities from the list under the numb variable.

```
for num in numb:  
    btn = Button(text=str(num),font_size = "30px", background_color =  
    (1.0,2.0,3.0,0.6)) btn.bind(on_release=self.cancel_numbs)  
    self.numbers.add_widget(btn)
```

```
for num in numbc:  
    btn = Button(text=str(num),font_size = "30px", background_color =  
    (1.0,0.0,0.0,1.0)) btn.bind(on_release=self.cancel_numbs)  
    self.numbers.add_widget(btn)
```

```
eq = Button(text="=", font_size = "30px", background_color =  
    (1.0,0.0,0.0,1.0)) eq.bind(on_release = self.eval_numb)  
self.numbers.add_widget(eq)
```

We use a for-loop that for a number in our numbers list, our buttons should have a font size of 30 pixels and background color blue. We then bind our buttons to another function. **def** cancel_numbs(self,instance): inputt = self.ids.inputt
inputt.text += instance.text
if instance.text == "**CA**": inputt.text = ""
elif instance.text == "**C**": inputt.text = inputt.text[:2]

We create a function for our cancel and cancel all buttons. First we tell the program that an input should be added to an existing text when pressed (inputt.text += instance.text). Then we use a conditional statement to tell the program that if "CA" is pressed it should return an empty text ("") on the screen, this means that all will be cancelled in the screen.

We use another conditional statement that if “C” is pressed we will subtract two texts from the screen. We say two and not one because when we click “C” it also added so we subtract two so we include the “C” and a number that we are actually cancelling.

```
def eval_numb(self,text): inputt = self.ids.inputt exp =inputt.text
eva = eval(exp)
inputt.text = str(eva)
```

We now create a function that tells the program to evaluate what ever has been inputted in mathematical forms and return to us the solution/answer.

```
class CalcApp(App):
```

```
def build(self):
```

```
return CalcWindow() if __name__=="__main__": CalcApp().run()
```

By now, you should be familiar with this class and the last conditional statement. This is to inherit our imported app module and return our window class so our app is built by the windows class.

. Let us now take a look at the kivy file and explain it

```
<CalcWindow>:
```

```
orientation:"vertical" canvas.before:
```

```
Rectangle:
```

```
pos: self.pos size: self.size source: "pic.jpg"
```

We start by telling the program that the file is for our **CalcWindow** class, this is so the program knows where to direct all instructions.

We make the orientation of our window vertical and use the canvas to tell the program that we want a rectangle on our window that will take in the position of the window and the size of the window (this means it will fully be around the window). We then import a background picture. Note that the picture should already be in your project file or your directory would be longer. To make things simpler, copy your image and paste it on the project file you have created.

```
BoxLayout:
```

```
id: textinput
```

```
size_hint_y: .1
```

```
TextInput:
```

```
id: inputt
```

```
size_hint_y:
```

```
padding: 10
```

```
font_size: 17
```

```
background_color: (1.0,1.0,1.0,1.0) focus: True
```

We then create three box layouts and give instructions within and give them ids so we later use them to call those instructions. Our box layout is divided into a 100% box, so the **size_hint_y = .1** tells the program that this box layout will only take in 10% of the space. The other took 70% and the last took 20%

```
BoxLayout:
```

```
id: main_btn size_hint_y: .8
```

```
GridLayout: id: numbers cols:4
```

```
spacing: 7
```

We gave a grid layout to our second box layout because we needed columns so each button is in its own box. We then gave them a spacing of seven so they are apart from each other. Your final app should now be returned and working well. Congratulations! You have built your first mobile app. Now save the code in a folder as we will return later to package our application and deploy it to playstore.

Section 5.4: Clock & Stopwatch

Let us now create a clock and stop watch app. This app will let us be able to view time and to use the stopwatch for competition purposes like running, coding or studying marathons.

Let us start by planning our application:

1. We want our clock to display 4 things: Time, A stopwatch, a start/stop button and a reset button.
2. We want our stopwatch not to start until we press start and not to

stop until we press stop.

3. We want our stopwatch to reset to zero when we click reset.

4. We want to add background images to our app background, start/stop button and reset button. This is to make our app more user friendly and the UI more beautiful.

5. We want our python file to be able to pull instructions from the kivy file.

6. We want our app to work□.

Main.py file

```
import kivy
from kivy.app import App
from kivy.core.text import LabelBase from kivy.core.window import
Window from kivy.utils import get_color_from_hex from kivy.clock
import Clock
from time import strftime

class StopWatch(App): starting_seconds = 0 watch_started = False
Window.size = (300,500)

def on_start(self):
    Clock.schedule_interval(self.update, 0)
    def update(self, nap):
        if self.watch_started:
            self.starting_seconds += nap self.root.ids.time.text =
            strftime('[b]%H[/b]:%M:%S')
            m, s = divmod(self.starting_seconds, 60)
            self.root.ids.stopwatch.text = ('%02d:%02d.[size=40]%02d[/size]' %
            (int(m), int(s), int(s * 100 % 100)))

    def start_stop(self):
        self.root.ids.start_stop.text = 'Start' if self.watch_started else 'Stop'
        self.watch_started = not self.watch_started

    def reset_button(self):
        if self.watch_started:
```

```
self.root.ids.start_stop.text = 'Start' self.watch_started = False
```

```
self.starting_seconds = 0
```

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

StopWatch.kv

```
<Label>:
```

```
canvas.before:
```

```
Rectangle:
```

```
pos: self.pos size: self.size source: "pic.jpg"
```

```
font_size: 60
```

```
markup: True
```

```
source: "pic.jpg"
```

```
<Mybuttons@Button>:
```

```
font_size: 25
```

```
bold: True
```

```
background_normal: 'green.png' background_down: 'green.png'
```

```
Label:
```

```
id: time
```

```
text: '[b]00[/b]:00:00'
```

```
BoxLayout:
```

```
height: 90
```

```
orientation: 'horizontal' padding: 20
```

```
spacing: 20
```

```
size_hint: (1, None)
```

```
Mybuttons:
```

```
id: start_stop
```

```
text: 'Start'
```

```
on_press: app.start_stop()
```

```
Mybuttons:  
id: reset_button  
text: 'Reset'  
on_press: app.reset_button() background_normal: 'pink.png'  
background_down: 'pink.png'
```

```
Label:  
id: stopwatch  
text: '00:00.[size=40]00[/size]'
```

```
BoxLayout:  
id: title  
size_hint_y: .3  
Label:
```

```
text: "Musa's StopWatch" font_size: 30  
bold : True
```

We created two files and wrote the codes. On the python file, we wrote the structure of the code and on the kivy file, we wrote the design on the code (just like html and css). Now let us look the code in detail and explain it.

```
import kivy  
from kivy.app import App  
from kivy.core.text import LabelBase from kivy.core.window import  
Window from kivy.utils import get_color_from_hex from kivy.clock  
import Clock  
from time import strftime
```

As usual, we started by importing what we will need.

```
class StopWatch(App): starting_seconds = 0 watch_started = False  
Window.size = (300,500)
```

We then created a class where our functions will be stored and inherited the App module as our subclass. Our stopwatch will start at zero seconds and we put an automatically start at false, this means

our stopwatch will not start automatically until we press the start button. We then created the window size of our app.

```
def on_start(self):
```

```
    Clock.schedule_interval(self.update, 0)
```

We then created a function let lets our app start from zero and increase with split seconds interval from the clock module

```
def update(self, nap):
```

```
if self.watch_started:
```

```
    self.starting_seconds += nap self.root.ids.time.text =
```

```
    strftime('[b]%H[/b]:%M:%S')
```

```
    m, s = divmod(self.starting_seconds, 60)
```

```
    self.root.ids.stopwatch.text = ('%02d:%02d.[size=40]%02d[/size]' %  
    (int(m), int(s), int(s * 100 % 100)))
```

Here we define a function on how we want our time displayed and updated. In chapter 2.1 you were given a table of what each operator means interms of how we want times and dates displayed, so I will not discuss that in detail. The **divmod()** method takes two parameters x and y, where x is treated as numerator and y is treated as denominator. To get our split seconds, we used mathematical operators and we let our split seconds run to a hundred.

```
def start_stop(self):
```

```
    self.root.ids.start_stop.text = 'Start' if self.watch_started else 'Stop'
```

```
    self.watch_started = not self.watch_started
```

```
def reset_button(self):
```

```
if self.watch_started:
```

```
    self.root.ids.start_stop.text = 'Start' self.watch_started = False
```

```
    self.starting_seconds = 0
```

We then used conditional staements what our start/stop button should display depending on when it has been pressed. Initially it will display start and once pressed it will display stop and stop the clock. The interchanging will keep repeating until you press reset, which resets the stopwatch to zero.

```
<Label>:  
canvas.before:  
Rectangle:
```

```
pos: self.pos size: self.size source: "pic.jpg"
```

```
font_size: 60  
markup: True  
source: "pic.jpg"
```

We then created our kivy file and added a background image of the screen and two background images of our buttons. We gave our image the size and position of the screen so it fits perfectly. We added two box layouts for, one for our time screen, one for our names. After running the app, your stopwatch should be returned.

Section 5.5: A Chat Server

To develop a chat server, we begin by developing the server-side code so that we have an endpoint to connect to before we begin writing the client. We will use the **Twisted** framework, which reduces many common, low-level networking tasks to a small number of clean python code.

To install Twisted, go to your command prompt and type in:
pip install -U twisted

Then set it up by going to your project in pycharm, clicking on file, settings, project interpreted, the plus icon and search for twisted then install the package (Just like how you set up your kivy in the beginning.). Now create a new python file and call it server.py and type in the code below:

```
from twisted.internet import protocol, reactor  
transports = set()  
class Chat(protocol.Protocol): def dataReceived(self, data):  
transports.add(self.transport) if ':' not in data: return
```

user, msg = data.split(':', 1) Server.py



```
for t in transports:
    if t is not self.transport:
        t.write('{0} says: {1}'.format(user, msg))
class ChatFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Chat()
reactor.listenTCP(9096, ChatFactory())
reactor.run()
```

Our server works this way:

- **reactor.run()**, starts the **ChatFactory** server that listens on port 9096.

- When the server receives input, it invokes the **dataReceived()** callback.

- The **dataReceived()** method implements the pseudocode from the protocol section, sending messages to other connected clients as required

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm (- wiki).

In this case; below is our pseudocode: if ':' not in message then

```
// it's a CONNECT message
add this connection to user list else
```

```
// it's a chat message
```

```
nickname, text := message.split on ':' for each user in user list
```

```
if not the same user:
```

```
send "{nickname} said: {text}"
```

After writing the full code and you run the application, it will should not return anything. The server just waits for the clients to connect.

Before we move on into our building our GUI, we need to learn about databases so we are able to save our text inputs, messages, images

to databases. We will end here for now and cover the next chapter on SQLite (database) and later to finish our server when we build a chat application.

Chapter 6: Introduction to SQLite

SQLite is a relational database management system. The reason I have chosen SQLite is that it is already built into python; so you do not need to install or set up anything, you will just import it whenever you want to use it. It is also the easiest database language I recommend to be used.

To get started, create a new python file and call it **database.py**; this is the file we will be using for our database. To use SQLite in our python file, we start by importing sqlite3, which is the database we will be using.

```
import sqlite3
```

We then want to create our database and connect it to our python file. To do this we use a connect() method.

As an example, we will be using a school class so we can create a database for them:

```
import sqlite3
```

```
grade_12 = sqlite3.connect("Grade 12 class")
```

We create a variable (grade_12), assign in to our sqlite3, and connect it to our database (Grade 1 Class). Since this database does not exist, the computer will automatically create one for us. If you run this program, it will return nothing, but you will see that a file for your database will appear under your project files. Our database will still be empty (Nothing will be in the database file).

Now let us create our table in our database that will contain the information if the learners, for instance the name, surname and stream (Science, Commerce, Humanities).

To create a table, you first need to create a cursor and to do this; we create a variable for it. We use a cursor to do anything.

When we create a table, we always need to indicate what type of data will be store in there. In sqlite3, we only have five data types:

NULL: means does not exist INTEGER: a whole number (no decimal)
REAL: floats (has decimal) TEXT: text BLOB: images, videos, images
and any other stored thing. **import** sqlite3

```
grade_12 = sqlite3.connect("Grade 12 class")  
c = grade_12.cursor()
```

```
c.execute( """CREATE TABLE learners( first_name TEXT,  
last_name TEXT,  
stream TEXT,  
  
)""")  
grade_12.commit()
```

We create our database using the structure above. If you run this program and open your database file again, your table will now be created. Note that sqlite3 is case sensitive so its methods must always be in capital letters. We the end the code by committing our variable, this is a necessary step in every database.

Let us now start adding the information of students in our database:

```
import sqlite3  
grade_12 = sqlite3.connect("Grade 12 class")  
c = grade_12.cursor()  
c.execute("INSERT INTO learners VALUES ('Simon', 'Naidoo',  
'Sciences')")  
grade_12.commit()
```

We use the INSERT INTO method to insert our code into our database. Note that inside the value brackets I used single quotation marks and not double because we already used the double to put in the insert method. When you run the program and go to your database file, you will see that this learner has been added. To add another learner, simply remove that information, enter new

information, and run the program and now two learners will be in the database.

Let us enter five more learners and their information so we have some data to use from our database:

Learner Two:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("INSERT INTO learners VALUES ('Caleb', 'Levy', 'Commerce')")
grade_12.commit()
```

Learner Three:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("INSERT INTO learners VALUES ('Michelle', 'Ethan', 'Sciences')")
grade_12.commit()
```

Learner Four:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("INSERT INTO learners VALUES ('Letticia', 'Ndlovu', 'Humanities')")
```

Learner 5:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("INSERT INTO learners VALUES ('Gashwin', 'Naidoo', 'Commerce')")
grade_12.commit()
```

We have now added 5 learners to our database. Let us now print all the learners and their information to display when we run the code:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("SELECT * FROM learners")
```

```
print(c.fetchall())  
grade_12.commit()
```

This will return all your learners and their information back to the console.

Alternatively, if you only want to print out the first learners information you can instead use fetchone():

```
import sqlite3  
grade_12 = sqlite3.connect("Grade 12 class")  
c = grade_12.cursor()  
c.execute("SELECT * FROM learners")
```

`print(c.fetchone())` Sqlite3 also allows you to fetch only one item in the list. We will still use fetch one, but reference the list:

```
import sqlite3  
grade_12 = sqlite3.connect("Grade 12 class")  
c = grade_12.cursor()  
c.execute("SELECT * FROM learners")  
print(c.fetchone()[0])  
grade_12.commit()
```

This will return only the first learners name. This uses the same concept we covered of accessing list in the first two chapters on this book.

We can also assign fetchall() to a variable and then instead print the variable:

```
import sqlite3  
grade_12 = sqlite3.connect("Grade 12 class")  
c = grade_12.cursor()  
c.execute("SELECT * FROM learners")  
my_learners = c.fetchall()  
print(my_learners)  
grade_12.commit()
```

We create a variable called my_learners and assign it to c.fetchall() and the print out the variable. This usually helps us in a later stage when we want to use the information with some else.

```
Example: c = grade_12.cursor()  
c.execute("SELECT * FROM learners")  
my_learners = c.fetchall()
```

```
print(" The grade 12 students are {}".format(my_learners))
grade_12.commit()
```

Here we use the format method to use the variable.

We still have a problem; all our learners are printed in one line, making it clumsy. To fix this we use the for loop method in order to print the list out properly:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("SELECT * FROM learners")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This will return our learners in a better way.

Now our list is readable, we can further customize our code; an example being if we only want our list to be only the names of students and not their full information:

```
c = grade_12.cursor() c.execute("SELECT * FROM learners")
my_learners = c.fetchall()
for learner in my_learners: print(learner[0])
grade_12.commit()
```

This will only return the names of our learners.

Primary keys

Primary keys are unique row ids for every record of information saved. Sqlite3 always automatically creates this for us but does not show them when we print unless we specify. Primary keys can never be repeated. Let us print out our information and make the primary keys visible:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("SELECT rowid, * FROM learners")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This should now return the information with their primary keys, here in would be 1-5. Primary keys are useful when we build apps that require users; we can always see how many users our database now has, etc.

We can also query our database for certain conditions using the where clause. This is like the conditional statement for databases. Let us print the list of the learners where the last name is "Naidoo":

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("SELECT * FROM learners WHERE last_name =
'Naidoo' ")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This should return only the learner with a last name of Naidoo. If a database does not have that surname, it would return nothing. If I have not explained; selecting * means selecting everything.

We can also use the like method if we want say the names that start with the letter L in our database:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("SELECT * FROM learners WHERE first_name LIKE
'L%' ")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This should return all the names that start with the letter L when we run our program. We can also update our records using the update method on our database. Let us say Leticia has changes her name to Candice, let us update that. First, we would need to know her row id and you could get it by re printing the list of students with the row id:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
```

```
c = grade_12.cursor()
c.execute("SELECT rowid, * FROM learners")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This will return the list of learners and Leticia having row id 4, now let us change her name:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("""UPDATE learners SET first_name = 'Candice'
WHERE rowid = 4
""")
```

```
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This will return nothing when you run it, but the database has been updated. To be sure, let us go ahead and reprint the list of our learners again:

```
c = grade_12.cursor() c.execute("SELECT rowid, * FROM
learners")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

Leticia, with row id four now has the name Candice.

You can delete a record from a table by using the delete method. Let us delete the fifth learner:

```
import sqlite3
grade_12 = sqlite3.connect("Grade 12 class")
c = grade_12.cursor()
c.execute("DELETE FROM learners WHERE rowid = 5")
my_learners = c.fetchall()
for learner in my_learners: print(learner)
grade_12.commit()
```

This will delete the learner with row id = 5 and once you delete a record you can never get it again, so always be careful when using the delete method.

To verify that the learner with row id 5 has been deleted, let us go ahead and print the list of all learners again:

```
c = grade_12.cursor() c.execute("SELECT rowid, * FROM learners")
```

```
my_learners = c.fetchall()
```

```
for learner in my_learners: print(learner)
```

```
grade_12.commit()
```

When we run this, the learner with row id 5 will no longer exist.

We can also delete the whole table by using the drop method.

Deleting a table means deleting everything inside it so when building apps one should always be careful:

```
import sqlite3
```

```
grade_12 = sqlite3.connect("Grade 12 class")
```

```
c = grade_12.cursor()
```

```
c.execute("DROP TABLE learners")
```

```
my_learners = c.fetchall()
```

```
for learner in my_learners: print(learner)
```

```
grade_12.commit()
```

This will delete our table and our table will no longer exist.

Chapter 7: Complex Mobile Apps

At this point we have covered all the basics one needs to put together to build mobile apps. Sometimes the app you might have an idea of might already exist, so it saves time for you to just use their source code as reference; to do this, you need to be able to read and understand code. That is the importance of this chapter. This chapter will be the longest, as the code to complex apps will take many pages.

After this chapter, you will be ready to start building your own mobile apps.

Section 7.1: Bricky Math

The app is a math drag and drop app that was originally created by Jan Kaliszewski and took 3rd place at the 2014 kivy contest. It allows the users to drag and drop boxes of math equations to get a solution for the math equation. The full code is below:

Main.py

```
from __future__ import division, unicode_literals

import collections import functools import operator import random

import kivy
from kivy.config import Config

from kivy.app import App
from kivy.animation import Animation from kivy.clock import Clock
from kivy.core.audio import SoundLoader from kivy.properties
import (

    AliasProperty,
    BooleanProperty,
    NumericProperty,
    ListProperty,
    ObjectProperty,
    OptionProperty,
    ReferenceListProperty,

)
from kivy.uix.behaviors import DragBehavior from kivy.uix.label
import Label
from kivy.uix.popup import Popup
from kivy.uix.widget import Widget
from kivy.utils import interpolate, platform from kivy.vector import
Vector

SYMBOL_TO_BRICK_TEXT = {
    '==': '=',
    '+': '+',
    '-': '\u2212',
```

```

'*': '\xd7',
'/': '\xf7',

}
BRICK_TEXT_TO_SYMBOL = {
text: symbol for symbol, text in SYMBOL_TO_BRICK_TEXT.items()}

DIFFICULTY_LEVEL_LIMITS = [ dict(
equalities=1,
ops='+',
min_number=1,
max_number=4,
max_total_number=8,
max_symbols_per_equality=6,

),

dict(
equalities=1,
ops='-',
min_number=1,
max_number=8,
max_total_number=8,
max_symbols_per_equality=6,

),

dict(
equalities=1,
ops='+-',
min_number=0,
max_number=8,
max_total_number=10,
max_symbols_per_equality=7, ops='*',
min_number=0,
max_number=4,
max_total_number=9,
max_symbols_per_equality=6,

```

```
),
dict(
equalities=1,
ops='/',
min_number=0,
max_number=9,
max_total_number=9,
max_symbols_per_equality=6, ),
dict(
equalities=1,
ops='+-*',
min_number=2,
max_number=10,
max_total_number=12,
max_symbols_per_equality=8, ),

dict(
equalities=1,
ops='+-/',
min_number=0,
max_number=10,
max_total_number=12,
max_symbols_per_equality=9,

),
dict(
equalities=1,
ops='+-*/',
min_number=0,
max_number=10,
max_total_number=12,
max_symbols_per_equality=10, ops='+-*/',
min_number=0,
max_number=10,
max_total_number=20,
max_symbols_per_equality=11, ),
dict(
equalities=1,
```

```

ops='+-*/',
min_number=0,
max_number=10,
max_total_number=50,
max_symbols_per_equality=12, ),
dict(
equalities=1,
ops='+-*/',
min_number=0,
max_number=10,
max_total_number=100,
max_symbols_per_equality=13, ),
dict(
equalities=2,
ops='+-*/',
min_number=0,
max_number=10,
max_total_number=40,
max_symbols_per_equality=8, ),

dict(
equalities=2,
ops='+-*/',
min_number=0,
max_number=30,
max_total_number=100,
max_symbols_per_equality=9, ops='+-*/',
min_number=0,
max_number=40,
max_total_number=1000, max_symbols_per_equality=10,

),
dict(
equalities=3,
ops='+-*/',
min_number=0,
max_number=50,
max_total_number=5000,

```

```
max_symbols_per_equality=12, ),  
]
```

```
MAX_RETRY = 40
```

```
MIN_REPEATING_SYMBOL_COMBINATION_INTERVAL = 4
```

```
SOUND_FILENAME_PATTERN = 'sounds/arithmebricks-  
{0}_Seq01.wav' SOUND_ID_TO_SYMBOL = {
```

```
'eq': '==',
```

```
'add': '+',
```

```
'sub': '-',
```

```
'mul': '*',
```

```
'div': '/',
```

```
}
```

```
HELP_TEXT = (  
'Drag and drop the bricks (digits and operators) ' 'to form valid
```

```
equalities (e.g. [i]2+10=15-3[/i]).\n' 'All given bricks must be used.  
,
```

```
'There is always at least one valid solution.'
```

```
)
```

```
class ArithmeBricksApp(App):
```

```
def build(self):
```

```
self.icon = 'icon.png'
```

```
self.load_sounds()
```

```
game = ArithmeBricksGame()
```

```
Clock.schedule_once(lambda dt: game.show_title(), 1) return game
```

```
def load_sounds(self):
```

```
self.symbol_to_sound = {}
```

```
sound_ids = list('0123456789') + list(SOUND_ID_TO_SYMBOL) for
```

```
sound_id in sound_ids:
```

```
filename = SOUND_FILENAME_PATTERN.format(sound_id) symbol
```

```
= SOUND_ID_TO_SYMBOL.get(sound_id, sound_id)
```

```
self.symbol_to_sound[symbol] = SoundLoader.load(filename)
```

```
def play_sound(self, symbol, delay=None, volume=0.15): sound =  
self.symbol_to_sound.get(symbol)  
if sound is not None:
```

```
def callback(dt):  
    sound.volume = volume  
    sound.play()
```

```
if delay is None:  
    delay = random.randint(0, 20) / 50  
    Clock.schedule_once(callback, delay)
```

```
class ArithmeBricksGame(Widget):  
    difficulty_level_limits = DIFFICULTY_LEVEL_LIMITS  
    playing = BooleanProperty(False) finished = BooleanProperty(False)
```

```
    limits = ObjectProperty()  
    width_brick_ratio = NumericProperty() min_width_brick_ratio =  
    NumericProperty() brick_width = NumericProperty()  
    brick_height = NumericProperty()  
    panel_height = NumericProperty()  
    aux_text_size = NumericProperty()
```

```
    title_lines = ListProperty()
```

```
def __init__(self, *args, **kwargs):  
    super(ArithmeBricksGame, self).__init__(*args, **kwargs)  
    self.symbol_generator = SymbolGenerator()
```

```
def new_game(self):  
    self.playing = self.finished = False self.clear_bricks()  
    self.provide_bricks()  
    self.playing = True
```

```
def clear_bricks(self):  
for brick in list(self.iter_all_bricks()): Animation.cancel_all(brick)  
    self.remove_widget(brick)
```

```

def provide_bricks(self):
    limits = self.limits
    self.width_brick_ratio = max(

self.min_width_brick_ratio,
limits['max_symbols_per_equality']) + limits['equalities'] 1 for
symbol in self.symbol_generator(limits):
    self.add_new_brick(symbol)

def add_new_brick(self, symbol):
    target_pos = self.new_pos()
    if symbol in SYMBOL_TO_BRICK_TEXT:

    if symbol == '==':
        brick = EqualityBrick()
    else:
        brick = OperatorBrick()
    brick.text = SYMBOL_TO_BRICK_TEXT[symbol] else:
        assert symbol in '0123456789'
        brick = DigitBrick()
    brick.text = symbol
    self.add_widget(brick)
    brick.pos = self.center
    brick.target_pos = target_pos

def new_pos(self):
    for i in range(MAX_RETRY * 2):
        x = random.randint(5, self.width 5 - int(self.brick_width)) y =
        random.randint(5 + int(self.brick_height),

self.height - int(self.brick_height))
        min_distance = self.brick_width
        _distance = Vector(x, y).distance
        if all(_distance(brick.target_pos) >= min_distance

    for brick in self.iter_all_bricks()):
        break
    return x, y

```

```

def iter_all_bricks(self):
    return (obj for obj in self.children if isinstance(obj, Brick))

def finish_game(self):
    if self.playing:
        self.finished = True
        self.playing = False
    def popup_help(self):
        HelpPopup().open()
    def popup_quit(self):
        QuitPopup().open()
    def popup_new_game(self):
        def on_dismiss(popup):
            if popup.user_decision:
                self.new_game()
        NewGamePopup(on_dismiss=on_dismiss).open()

def show_title(self):
    mid_row = len(self.title_lines) / 2
    for row, line_text in enumerate(self.title_lines):

        Clock.schedule_once(
            functools.partial(
                self.show_title_row,
                mid_row,
                row,
                line_text,
            ),
            row * 0.3)
    def show_title_row(self, mid_row, row, line_text, dt):
        if self.playing:
            return
        mid_col = len(line_text) / 2
        for col, char in enumerate(line_text):
            if char == ' ':
                continue
            pos = self.new_pos()
            brick = TitleBrick()
            self.add_widget(brick)
            brick.text = char

```



```

brick.pos = pos
brick.target_pos = (

self.center_x + (col - mid_col) * self.brick_width, self.center_y - (row -
mid_row) * self.brick_height self.brick_height / 2)
class Brick(DragBehavior, Label):

# NOTE: values of properties without defaults # shall be set in the .kv
file
background_color = ListProperty()
border_color = ListProperty()

# (declaring the following class-wide constants as properties makes it
# easier to specify/inherit/override their values just in the .kv file)
detached_border_color = ListProperty()
move_border_color = ListProperty()
attached_border_color = ListProperty()
equal_border_color = ListProperty()
final_border_color = ListProperty()

max_snap_x_distance = NumericProperty() max_snap_y_distance =
NumericProperty() max_double_attach_x_distance =
NumericProperty() max_double_attach_y_distance =
NumericProperty()

target_x = NumericProperty(0)
target_y = NumericProperty(0)
target_pos = ReferenceListProperty(target_x, target_y)

def get_target_right(self):
return self.target_x + self.width
def set_target_right(self, value):
self.target_x = value - self.width
target_right = AliasProperty(
get_target_right, set_target_right, bind=('target_x', 'width'))

target_right_pos = ReferenceListProperty(target_right, target_y) state
= OptionProperty('detached', options=[ 'detached',

```

```

'move',
'attached',
'equal',
'final',

])
left_attached_brick = None right_attached_brick = None
@property
def symbol(self): return self.text # event dispatch

def on_touch_down(self, touch):
if self.state != 'final' and super(Brick, self).on_touch_down(touch):
self.update_states_before_detach()
self.detach()
self.state = 'move'
return True

return False

def on_touch_up(self, touch):
if self.state != 'final' and super(Brick, self).on_touch_up(touch):
self.target_pos = self.pos
assert self.state == 'move'
if self.attach():

self.update_states_after_attach()
else:
self.state = 'detached'
return True

def update_states_before_detach(self): for brick_seq in
(self.collect_all_left(), self.collect_all_right()): if brick_seq:
if len(brick_seq) == 1:
brick_seq[0].state = 'detached' elif
self.is_brick_seq_equal(brick_seq): for brick in brick_seq:
brick.state = 'equal'
else:

```

```
for brick in brick_seq:
    brick.state = 'attached'
```

```
def detach(self):
    left_brick = self.left_attached_brick if left_brick is not None:
```

```
    left_brick.right_attached_brick = None self.left_attached_brick = None
    right_brick = self.right_attached_brick
    if right_brick is not None:
        right_brick.left_attached_brick = None self.right_attached_brick =
        None
```

```
# attaching
```

```
def attach(self):
    (left_brick,
     right_brick,
     target_pos) = self.get_left_right_bricks_and_target_pos() if left_brick
    is not None:
```

```
    left_brick.right_attached_brick = self.proxy_ref
    self.left_attached_brick = left_brick.proxy_ref if right_brick is not
    None:
    right_brick.left_attached_brick = self.proxy_ref
    self.right_attached_brick = right_brick.proxy_ref if target_pos is not
    None:
    self.target_pos = target_pos
    return left_brick is not None or right_brick is not None
```

```
def get_left_right_bricks_and_target_pos(self):
    left_brick = self.choose_left_brick()
    right_brick = self.choose_right_brick()
    if right_brick is not None:
```

```
        target_pos_by_right = (right_brick.target_x - self.width,
                                right_brick.target_y)
        if left_brick is not None:
            target_pos_by_left = tuple(left_brick.target_right_pos)
```

```

distance_from_left = (Vector(self.target_pos)
.distance(left_brick.target_right_pos)) distance_from_right =
(Vector(self.target_right_pos)

.distance(right_brick.target_pos)) if self.can_attach_to_both(
left_brick, right_brick,
target_pos_by_left, target_pos_by_right, distance_from_left,
distance_from_right):

target_pos = interpolate(target_pos_by_left, target_pos_by_right,
step=2)

elif self.should_attach_to_left(
left_brick, right_brick,
distance_from_left, distance_from_right):

target_pos = target_pos_by_left
right_brick = None
else:
target_pos = target_pos_by_right
left_brick = None
else:
target_pos = target_pos_by_right
elif left_brick is not None:
target_pos = left_brick.target_right_pos

else:
target_pos = None
return left_brick, right_brick, target_pos

def choose_left_brick(self):
_distance = Vector(self.target_pos).distance bricks_and_distances = [

(brick,
_distance(brick.target_right_pos),
abs(self.target_x - brick.target_right))
for brick in self.iter_all_bricks()
if brick.right_attached_brick is None]

```

```
return self.get_attachable_brick(bricks_and_distances)
```

```
def choose_right_brick(self):  
    _distance = Vector(self.target_right_pos).distance  
    bricks_and_distances = [  

```

```
(brick,  
    _distance(brick.target_pos),  
    abs(self.target_right - brick.target_x))  
    for brick in self.iter_all_bricks()  
    if brick.left_attached_brick is None]
```

```
return self.get_attachable_brick(bricks_and_distances)
```

```
def get_attachable_brick(self, bricks_and_distances):  
    bricks_and_distances.sort(key=operator.itemgetter(1))  
    for brick, _,  
    x_distance in bricks_and_distances:
```

```
    if brick == self:  
        continue  
    # (for checking snap limits, using x and y separately  
    # plays better than using the real x*y distance)  
    y_distance = abs(self.target_y - brick.target_y)  
    if (x_distance > self.max_snap_x_distance or y_distance >  
        self.max_snap_y_distance): return None  
    if self.can_be_attached_to(brick):  
        return brick
```

```
def can_be_attached_to(self, brick): return brick.state != 'move'
```

```
def can_attach_to_both(self, left_brick, right_brick,  
    target_pos_by_left, target_pos_by_right,  
    distance_from_left, distance_from_right):
```

```
    return (Vector(target_pos_by_left).distance(target_pos_by_right) <  
        self.width / 3) or (  

```

```
(distance_from_right / 3.5 <=  
    distance_from_left <=
```

3.5 * distance_from_right) and

(for checking snap limits, using x and y separately # plays better than using the real x*y distance)

(abs(self.target_x - left_brick.target_right) <=

self.max_double_attach_x_distance) and

(abs(self.target_y - left_brick.target_y) <=

self.max_double_attach_y_distance) and

(abs(self.target_right - right_brick.target_x) <=

self.max_double_attach_x_distance) and

(abs(self.target_y - right_brick.target_y) <=

self.max_double_attach_y_distance))

def should_attach_to_left(self, left_brick, right_brick,

distance_from_left, distance_from_right): # (for choosing the side,

comparing y distances often # seems to play better than comparing

real x*y distances) from_left = abs(self.target_y - left_brick.target_y)

from_right = abs(self.target_y - right_brick.target_y) if (from_left <

self.height / 4 and

from_right < self.height / 4) or (

from_right / 1.4 <=

from_left <=

1.4 * from_right):

y distances are too small or too similar to be # conclusive => let's

compare real x*y distances from_left = distance_from_left

from_right = distance_from_right

if from_left <= from_right:

return True

else:

assert from_left > from_right

return False

def update_states_after_attach(self):

brick_seq = self.collect_all_left()

brick_seq.append(self)

self.collect_all_right(brick_seq)

if self.is_brick_seq_equal(brick_seq):

```

for brick in brick_seq:
    brick.state = 'equal'
    all_bricks = list(self.iter_all_bricks())
    if all(brick.state == 'equal' for brick in all_bricks):
        for brick in all_bricks:
            brick.state = 'final'
        self.parent.finish_game()
    else:
        for brick in brick_seq:
            brick.state = 'attached'

```

commons

```

def collect_all_left(self, brick_seq=None):
    if brick_seq is None:
        brick_seq = []

```

```

    left_attached_brick = self.left_attached_brick
    if left_attached_brick is not None:

```

```

        left_attached_brick.collect_all_left(brick_seq)
        brick_seq.append(left_attached_brick)

```

```

    return brick_seq

```

```

def collect_all_right(self, brick_seq=None):
    if brick_seq is None:
        brick_seq = []

```

```

    right_attached_brick = self.right_attached_brick
    if right_attached_brick is not None:

```

```

        brick_seq.append(right_attached_brick)
        right_attached_brick.collect_all_right(brick_seq)
    return brick_seq

```

```

def is_brick_seq_equal(self, brick_seq):

```

```

    expr_str = ".join(brick.symbol for brick in brick_seq) if '==' not in expr_str:

```

```

    return False

```

```

    try:

```

```

        return eval(expr_str)

```

```

    except (SyntaxError, ArithmeticError):

```

```

        return False

```

```

def iter_all_bricks(self):
    return self.parent.iter_all_bricks()
class DigitBrick(Brick): pass
class OperatorBrick(Brick):
    @property
    def symbol(self):
        return BRICK_TEXT_TO_SYMBOL[self.text]
    def can_be_attached_to(self, obj):
        return (super(OperatorBrick, self).can_be_attached_to(obj) and
                isinstance(obj, DigitBrick))
class EqualityBrick(OperatorBrick): pass
class TitleBrick(Brick):
    def on_touch_down(self, touch): return False
    def on_touch_up(self, touch): return False
class HelpPopup(Popup): help_text = HELP_TEXT
class QuitPopup(Popup): pass
class NewGamePopup(Popup):
    user_decision = BooleanProperty(False)
#
# Helper classes
class SymbolGenerator(object):
    class _FailedToMakeEquality(Exception): pass
    def __init__(self):
        self.recent_symbol_combinations = collections.deque(
            maxlen=MIN_REPEATING_SYMBOL_COMBINATION_INTERVAL)

    def __call__(self, limits):
        vars(self).update(limits)
        while True:

            generated_symbols = list(self.generate_symbols())
            if not (self.are_too_easy(generated_symbols) or
                    self.repeated_too_soon(generated_symbols)):
                return iter(generated_symbols)

    def generate_symbols(self):
        equalities = self.equalities
        max_num_digits = len(str(self.max_number))

```



```

while True:
    left_max_symbols = random.randint(
        self.max_symbols_per_equality // 3,
        self.max_symbols_per_equality - 2)

    right_max_symbols = (
        self.max_symbols_per_equality
        - left_max_symbols
        - 1)

    try:
        equality, total_num = self.make_left_side(left_max_symbols,
            max_num_digits)

        equality.append('==')
        equality.extend(self.make_right_side(total_num, right_max_symbols,
            max_num_digits))
    except self._FailedToMakeEquality:
        continue

    assert '==' in equality and eval("".join(equality))
    for symbol in equality:
        yield symbol
    equalities -= 1
    if equalities < 1:
        break

def are_too_easy(self, generated_symbols):
    # eliminate symbol combinations that include too few symbols if
    len(generated_symbols) < self.max_symbols_per_equality - 3:

    return True
    ones = generated_symbols.count('1')
    # eliminate boring symbol combinations that include too many '1' if
    ones > max(2, len(generated_symbols) / (3 + self.equalities)): return
    True

# when there are few symbols: often (but not always) # eliminate lone
multiplications/divisions by 1 if ones and len(generated_symbols) <=

```

5:

```
symbol_set = set(generated_symbols)
if ((random.randint(1, 8) < 8 and
len(symbol_set.difference(('==', '*', '1')) == 1) or
```

```
(random.randint(1, 3) < 3 and
len(symbol_set.difference(('==', '/', '1')) == 1)):
return True
muls_and_divs = (generated_symbols.count('*') +
generated_symbols.count('/'))
```

```
# eliminate symbol combinations that include neither # '*' nor '/' when
any of that operators is available if ('*' in self.ops or '/' in self.ops) and
not muls_and_divs:
```

```
return True
```

```
# eliminate symbol combinations that are too easy because # of
possibility of tricks involving '0' combined with '*' # or '/' and arbitrary
digits (such as '2=2+0*17346348') zeros =
generated_symbols.count('0')
if not zeros:
```

```
return False
```

```
if not muls_and_divs:
```

```
return False
```

```
assert zeros > 0 and muls_and_divs
```

```
if zeros == 1 and not ('+' in generated_symbols or '-' in
generated_symbols):
```

```
return False
```

```
if ((zeros == 1 or muls_and_divs == 1) and
```

```
self.equalities == 1 and
```

```
random.randint(1, 5) == 5):
```

```
# sometimes we are lenient :) (but never on harder levels)
```

```
return False
```

```
return True
```

```

def repeated_too_soon(self, generated_symbols):
    symbol_combination = tuple(sorted(generated_symbols)) if
    symbol_combination in self.recent_symbol_combinations:

    return True
    self.recent_symbol_combinations.append(symbol_combination)
    return False

def make_left_side(self, cur_max_symbols, max_num_digits): num =
div_mul_operand = random.randint(self.min_number,
self.max_number)

symbols = list(str(num))
if len(symbols) > cur_max_symbols - 2:
    op = random.choice(self.ops)
    if op == '/':

num = self._random_divisor(div_mul_operand,
self.min_number,
self.max_number)

elif op == '*':
num = self._random_multiplier(div_mul_operand, self.min_number,
self.max_number,

self.max_total_number)
else:
num = random.randint(self.min_number, self.max_number)

draft_symbols = symbols[:]
draft_symbols.append(op)
draft_symbols.extend(str(num))
total_num = eval("".join(draft_symbols))
assert total_num == int(total_num)
if (total_num < self.min_number or

total_num > self.max_total_number or
len(draft_symbols) > cur_max_symbols):
    continue

```

```
div_mul_operand = self._new_div_mul_operand(op,  
div_mul_operand,  
num)
```

```
symbols = draft_symbols  
if len(symbols) > (cur_max_symbols  
max_num_digits  
random.randint(1, (self.equalities +  
max_num_digits  
1))):  
break  
else:  
raise self._FailedToMakeEquality  
return symbols, int(total_num)
```

```
def make_right_side(self, total_num, cur_max_symbols,  
max_num_digits): assert total_num <= self.max_total_number  
symbols = list(str(total_num))  
if len(symbols) > cur_max_symbols:  
if len(symbols) > (cur_max_symbols
```

```
max_num_digits  
1):  
return symbols  
for i in range(MAX_RETRY):
```

```
op = random.choice(self.ops)  
if op == '+':  
num2 = random.randint(self.min_number, self.max_number)  
num1 = total_num - num2  
elif op == '-':  
num2 = random.randint(self.min_number, self.max_number)  
num1 = total_num + num2  
elif op == '*':  
num2 = self._random_divisor(total_num,  
self.min_number,  
self.max_number)  
num1 = total_num // num2
```

```

assert num1 == total_num / num2
elif op == '/':
    num2 = self._random_multiplier(total_num,
    max(1, self.min_number),
    self.max_number,
    self.max_total_number)
    num1 = total_num * num2
    symbols = list(str(num1))
    symbols.append(op)
    symbols.extend(str(num2))
    if (self.min_number <= num1 <= self.max_total_number and
    self.min_number <= num2 <= self.max_number and len(symbols) <=
    cur_max_symbols):
        return symbols
    raise self._FailedToMakeEquality

```

```

@classmethod
def _random_divisor(cls, dividend, min_num, max_num): min_num =
max(1, min_num)
if random.randint(0, 40) != 40: # mostly avoid 1 min_num =
max(min_num, random.randint(2, 4))
max_num = min(dividend // 2 + 1, max_num)
if max_num < min_num:
    raise cls._FailedToMakeEquality
num = random.randint(min_num, max_num) if dividend % num == 0:
    break
else:
    max_num = min(10, max_num)
    if max_num < min_num:
        raise cls._FailedToMakeEquality
    for i in range(MAX_RETRY):
        num = random.randint(min_num, max_num)
        if dividend % num == 0:
            break
    else:
        raise cls._FailedToMakeEquality
    return num

```

```

@classmethod
def _random_multiplier(cls, multiplicand, min_num, max_num,
max_total_number):

if multiplicand != 0:
max_num = max_total_number // multiplicand
if random.randint(0, 40) != 40: # mostly avoid 0, often avoid 1...
min_num = max(min_num, random.randint(random.randint(1, 3), 4))
if max_num < min_num:
raise cls._FailedToMakeEquality
return random.randint(min_num, max_num)

```

```

@staticmethod
def _new_div_mul_operand(op, div_mul_operand, num): if op == '/':

return div_mul_operand / num
elif op == '*':
return div_mul_operand * num
else:
assert op in ('+', '-')
return num

```

Helper functions

```

def config_tweaks():
mouse_opt_str = Config.getdefault('input', 'mouse', None) if
mouse_opt_str:

mouse_options = mouse_opt_str.split(',')
# disable mouse-based multitouch emulation if 'disable_multitouch'
not in mouse_options: mouse_options.append('disable_multitouch')

# on linux: disable mouse when another multitouch device is used #
(see: http://kivy.org/docs/api-kivy.input.providers.mouse.html) if
platform == 'linux' and 'disable_on_activity' not in mouse_options:

mouse_options.append('disable_on_activity')
Config.set('input', 'mouse', ', '.join(mouse_options))

```

```
if __name__ in ('__main__', '__android__'): config_tweaks()
ArithmeBricksApp().run()
```

arithmebriks.kv:

```
#:kivy 1.8.0
```

```
#: import Animation kivy.animation.Animation #: import random
random
```

```
<ArithmeBricksGame>:
```

```
limits: self.difficulty_level_limits[int(difficulty_level_slider.value) - 1]
```

```
width_brick_ratio: 13 min_width_brick_ratio: 10
```

```
brick_width: self.width / (self.width_brick_ratio + 0.5)
```

```
brick_height: self.brick_width * 1.36
```

```
panel_height: self.width / 10.3
```

```
aux_text_size: (self.width * self.height) / (self.width + self.height) / 12.
```

```
title_lines: ['ArithmeBricks', '', 'by zuo']
```

```
background_color: 0.5, 0.65, 0.9, 1
```

```
Color:
```

```
rgba: self.background_color
```

```
Rectangle:
```

```
pos: self.pos
```

```
size: self.size
```

```
on_playing:
```

```
symbols = '0134' if args[1] else ''
```

```
for symbol in symbols: app.play_sound(symbol)
```

```
on_finished:
```

```
symbols = ['=='] + list('*-/+-0123456789') if args[1] else []
```

```
for i, symbol in enumerate(symbols): app.play_sound(symbol, delay=
(0.4 + i/10.), volume=0.1)
```

```
BoxLayout:
```

```
canvas.before:
```

Color:

rgba: 0, 0, 0.1, 1

Rectangle:

pos: self.pos

size: self.size

height: root.panel_height width: root.width

x: 0

y: 0

Button:

size_hint: (0.08, 1)

font_size: self.height / 2.8 text: 'Quit'

on_press:

root.popup_quit()

Button:

size_hint: (0.08, 1)

font_size: self.height / 2.8 text: 'Help'

on_press:

root.popup_help()

Label:

size_hint: (0.10, 1)

font_size: self.height / 2.8 align: 'center'

text: 'Level:'

Slider:

id: difficulty_level_slider

size_hint: (0.4, 1)

value: 1

min: 1

max: len(root.difficulty_level_limits)

step: 1

Label:

size_hint: (0.02, 1)

text: "

Button:

id: new_game_button

size_hint: (0.16, 1)

text: 'Play more' if root.finished else 'New Game'

color: (0.75, 0.9, 1, 1) if root.playing else (1, 1, 1, 1)

font_size: self.height / 2.8

on_press:

if root.playing: root.popup_new_game()

else: root.new_game()

<Brick>:

width: app.root.brick_width

height: app.root.brick_height

x: min(max(self.x, 0), app.root.width - self.width)

y: min(max(self.y, app.root.panel_height + 2), app.root.height - self.height)

bold: True

font_size: app.root.brick_height

drag_rectangle: self.x, self.y, self.width, self.height drag_timeout: 10000000

drag_distance: 0

background_color: 1, 1, 0.5, 1

detached_border_color: 0.2, 0.3, 0.55, 0.8 move_border_color: 0.6, 0.8, 1, 0.8 attached_border_color: 1, 0.9, 0.2, 0.5 equal_border_color: 0.1, 1, 0.5, 0.8 final_border_color: 0.2, 0.9, 0.2, 0.5

border_color: self.detached_border_color

max_snap_x_distance: 0.7 * self.width

max_snap_y_distance: 0.66 * self.height

max_double_attach_x_distance: self.max_snap_x_distance / 2

max_double_attach_y_distance: self.max_snap_y_distance / 3

Color:
rgba: self.background_color
Rectangle:
pos: self.pos
size: self.size

Color:
rgba: self.border_color
Line:
points: self.x, self.y, self.x + self.width, self.y, self.x + self.width, self.y
+ self.height, self.x, self.y + self.height, self.x, self.y width: 2

on_state:
cur_background_color = self.background_color
target_border_color = getattr(self, args[1] + '_border_color')
if args[1] == 'final': anim =
Animation(border_color=target_border_color,

background_color=self.equal_border_color, duration=0.2,
t='in_out_quad') +

Animation(border_color=self.equal_border_color,
background_color=cur_background_color,
duration=0.2, t='in_out_quad'); anim.repeat = True
else: anim = Animation(border_color=target_border_color,
duration=0.2, t='out_quint') anim.start(self)
if args[1] in ('attached', 'equal'): app.play_sound(self.symbol)

on_target_pos:
Animation(pos=args[1], duration=0.1, t='out_bounce').start(self)
<DigitBrick, TitleBrick>: color: 0.9, 0.3, 0.4, 1
<OperatorBrick>:
color: 0.2, 0.7, 0.4, 1

<EqualityBrick>:
color: 0.4, 1, 0.4, 1
background_color: 0.4, 0.55, 1, 1

<TitleBrick>:

font_size: app.root.brick_height * 0.67

on_target_pos:

if random.randint(1, 4) == 4: app.play_sound(random.choice('*-+368'), volume=0.15) <Button>:

color: 1, 1, 1, 1

background_color: 1, 1.5, 2, 1

<YesNoButton@Button>: size_hint: (0.4, 1)

<Popup>:

separator_color: app.root.background_color size_hint: None, None

title_size: app.root.aux_text_size

<HelpPopup>:

title: 'How to play'

size: app.root.width * 0.9, app.root.height * 0.9

BoxLayout:

orientation: 'vertical' padding: root.title_size spacing: root.title_size / 2

Label:

size_hint: (1, 0.8)

markup: True

text: root.help_text

text_size: (self.width - root.title_size, self.height - root.title_size)

font_size: root.title_size

halign: 'justify'

valign: 'top'

Button:

size_hint: (1, 0.2)

text: 'OK'

font_size: root.title_size

on_press:

root.dismiss()

<QuitPopup>:

```
title: 'Quit?'
size: app.root.width * 0.9, max(120, app.root.aux_text_size * 8)
```

```
BoxLayout:
orientation: 'horizontal' padding: root.title_size spacing: root.title_size
/ 2
```

```
YesNoButton:
text: 'Yes'
font_size: root.title_size on_press:
```

```
app.stop()
YesNoButton:
text: 'No'
```

```
font_size: root.title_size on_press:
root.dismiss()
```

```
<NewGamePopup>:
title: 'Abandon current game?'
size: app.root.width * 0.9, max(120, app.root.aux_text_size * 8)
user_decision: False
```

```
BoxLayout:
orientation: 'horizontal' padding: root.title_size spacing: root.title_size
/ 2
```

```
YesNoButton:
text: 'Yes'
font_size: root.title_size on_press:
```

```
root.user_decision = True root.dismiss()
```

```
YesNoButton:
text: 'No'
font_size: root.title_size on_press:
```

```
root.dismiss()
```

This should return your game. If you have any difficulties, you can get

the full source code at <https://github.com/zuo/ArithmeBricks.git>

Section 7.2: Hotel Management system

Let us now create a hotel management system app that will include a login database. To create this app you will need to again set up make you connect your database by importing it. This app will have main different python file and will be the most complex, so if you can't seem to understand it, you can go ahead and skip it as it needs lots of logic and understating of kivy. When coding, if you see any images attached, you can simply use your own images and just rename them to be the same as names in the code and put those images in the folder of your project:

Start.py

[Grab your reader's attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

```
from Screen import Login
from Screen.SignUp import SignUpScreen
from Screen.Section import SectionScreen
from Screen.Resturant import ResScreen
from Screen.Lodging import LodScreen

from sys import path
path.append('/usr/lib/python3/dist-packages/')
from kivy.uix.screenmanager import ScreenManager
from kivy.app import App
from kivy.clock import Clock

class SwitchScreen(ScreenManager):
    #Switch Screen is the main screen manager of the application
    def loginC(self):

        self.L = Login.LoginScreen()
        self.L.set()
        self.add_widget(self.L)
```

```
self.current = 'login'
self.SU = SignUpScreen()
self.SU.set()
self.add_widget(self.SU)
self.SE = SectionScreen()
self.SE.set()
self.add_widget(self.SE)
self.RS = ResScreen()
self.RS.set()
self.add_widget(self.RS)
self.LS = LodScreen()
self.LS.set()
self.add_widget(self.LS)
```

```
def update(self,dt):
# Checks if there is any change in the screen
if self.LS.bk == True or self.RS.rb.o.bk == True:
```

```
self.current = 'section'
self.SE.m.L.LSel = False
self.LS.bk = False
self.RS.rb.o.bk = False
self.SE.m.R.RSel = False
```

```
elif self.SE.m.R.RSel == True and self.L.B.LMB.LM.LoginT == True :
self.current = "CustResScreen"
elif self.SE.m.L.LSel == True:
self.current = "LodgingScreen"
elif self.L.B.LMB.LM.LoginT == True :
self.current = 'section'
elif self.L.B.LMB.LM.SignUpT == True :
self.current = 'signup'
if self.SU.X.SL.su.backtl == True or self.SU.X.SL.su.signupT == True:
self.L.B.LMB.LM.SignUpT = False
self.current = 'login'
self.SU.X.SL.su.backtl = False
self.SU.X.SL.su.signupT = False
```

```

class HotelManagementSystemApp(App):
# main application
def build (self):

self.a = SwitchScreen()
self.a.loginC()
Clock.schedule_interval(self.a.LS.update,1.0 / 10.0)
Clock.schedule_interval(self.a.update, 1.0 / 60.0)
Clock.schedule_interval(self.a.RS.rb.o.up,1.0 / 30.0) return self.a

```

```

if __name__ == '__main__':
HotelManagementSystemApp().run()

```

Login.py

```

from sys import path
path.append('/usr/lib/python3/dist-packages/')
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager,Screen
from kivy.uix.widget import Widget
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.anchorlayout import AnchorLayout
from kivy.modules import inspector
from kivy.core.window import Window
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.dropdown import DropDown
from kivy.uix.popup import Popup
#import mysql.connector as db
#hmdb =
db.connect(host="localhost",user="root",passwd="1997",db="hmsdb")
#cursor = hmdb.cursor()

#Builder.load_string("""
<LoginScreen>:

```

BoxLayout:

<Title>:

orientation: 'vertical'

size_hint : 1, .1

canvas:

Color :

rgba : .9411,.3843,.5725,1

Rectangle:

size : self.size

pos : self.pos

Label:

text : "Hotel Management System" color : 0,0,0,1

bold : True

font_size : 50

<LoginBg>:

canvas:

Rectangle:

source : 'b.jpg'

size : self.size

pos : self.pos

<LoginMenu>:

orientation : 'vertical' canvas:

Color :

rgba : 1,.8784,.5098,1

Rectangle:

size : self.size

pos : self.pos

<LoginMenuB>:

canvas:

Color:

rgba : 1,.8353,.3098,1

Rectangle :

size : self.size

pos : self.pos

<UP>:
orientation : 'horizontal'

<AccType>:

Button:
text : 'Customer'
size_hint_y : None
height : 36
on_release : root.select('Customer') background_normal: "
background_color : 1,.4392,.2627,1 color : 0,0,0,1

Button:
text : 'Employee'
size_hint_y : None
height : 36
on_release : root.select('Employee') background_normal: "
background_color : 1,.4392,.2627,1 color : 0,0,0,1

Button:
text : 'Manager'
size_hint_y : None
height : 36
on_release : root.select('Manager') background_normal: "
background_color : 1,.4392,.2627,1 color : 0,0,0,1

'''

class UP(BoxLayout):
Frame grouping label and input for login
def set(self):

self.padding = (5,5)
self.spacing = 10
def aset(self):

self.add_widget(Label(text = 'Account Type', color = (0,0,0,1)))
self.Account = AccType()

```
self.acc = Button(text = 'Select Acc. Type',background_normal=
'',background_color =
```

```
(1,.4392,.2627,1),color = (0,0,0,1))
self.acc.bind(on_release = self.Account.open)
self.Account.bind(on_select = lambda instance, x : setattr( self.acc
,'text', x)) self.add_widget(self.acc)
self.set()
```

```
def oset(self,t,f):
self.add_widget(Label(text = t + ' : ', color = (0,0,0,1)))
self.l = TextInput(hint_text = t, multiline = False, padding =
(10,10),password = f) self.add_widget(self.l)
self.set()
```

```
class Title(BoxLayout):
# Main title of the Application
def set(self):
```

```
self.size_hint = (1,.1)
self.pos_hint = {'top' : 1,'center_x' : 0.5}
class LoginBg(AnchorLayout):
```

```
#Background which consist of image background
def set(self):
self.size_hint = (1,1)
self.LMB = LoginMenuB()
self.LMB.set()
self.LoginT = self.LMB.LoginT
self.SignUpT = self.LMB.SignUpT
self.add_widget(self.LMB)
```

```
class LoginMenuB(BoxLayout):
# Square background of login menu
def set(self):
```

```
self.size_hint=(.37,.52)
self.pos_hint={'center_x' : 0.5,'center_y' : 0.5}
```

```

self.padding = (10,10)
self.LM = LoginMenu()
self.LM.set()
self.LoginT = self.LM.LoginT
self.SignUpT = self.LM.SignUpT
self.add_widget(self.LM)
def aw(self, ob):

self.add_widget(ob)

class PopUp(Popup):
# Show pop up if encounter error in the login
def set(self):

self.title = 'Wrong Login Details'
self.content = Label(text = 'You have entered \n wrong Login Details')
self.size_hint = (None,None)
self.size = (200,200)

class LoginMenu(BoxLayout):
# Login Menu with all the input and button widgets
def logincheck(self,instance):

self.query = "SELECT * FROM logindetails WHERE username = '{}'"
AND acctype='{}' ".format(self.UN.l.text,self.AS.acc.text)
#cursor.execute(self.query)
self.validcheck = cursor.fetchone()
if self.PW.l.text == "or self.validcheck == None:
self.p = PopUp()
self.p.set()
self.p.open()
else :
if self.PW.l.text == self.validcheck[1]:
self.LoginT = True
hmdb.close()
else :
self.p = PopUp()

```

```
self.p.set()  
self.p.open()
```

```
def signupcheck(self,instance): self.SignUpT = True
```

```
def set(self):  
self.LoginT = False  
self.SignUpT = False  
self.padding = (10,10)  
self.pos_hint={'center_x' : 0.5,'center_y' : 0.5}  
self.aw(Label(text='Login',color
```

```
self.AS = UP()  
self.AS.aset()  
self.aw(self.AS)  
self.UN = UP()  
self.UN.oset('Username',False)  
self.aw(self.UN)  
self.PW = UP()  
self.PW.oset('Password',True)  
self.aw(self.PW)  
self.lg = Button(text = 'Login',background_normal=  
",background_color =
```

```
(1,.4392,.2627,1),color = (0,0,0,1))  
self.lg.bind(on_press = self.logincheck)  
self.aw(self.lg)  
self.signup = BoxLayout(orientation = 'horizontal', padding =  
(5,5),spacing = 10) self.orl = Label(text = 'Or', size_hint = (.3,1),color  
= (0,0,0,1))  
self.signupb = Button(on_press = self.signupcheck,text = 'Sign Up',  
size_hint =
```

```
(.7,1),background_normal= ",background_color =  
(1,.4392,.2627,1),color = (0,0,0,1)) self.signup.add_widget(self.orl)  
self.signup.add_widget(self.signupb)  
self.aw(self.signup)
```

```

def aw(self,ob):
self.add_widget(ob)

class LoginScreen(Screen,BoxLayout): # main login screen
def set(self):

self.name = 'login'
self.orientation = 'vertical' self.T = Title()
self.T.set()
self.add_widget(self.T,index = 0) self.B = LoginBg()
self.B.set()
self.LoginT = self.B.LoginT self.SignUpT = self.B.SignUpT
self.add_widget(self.B,index = 1)

class AccType(DropDown): # To create drop down list pass

class Main():
# crete local login application def Start(self):

self.sm = ScreenManager() self.L = LoginScreen() self.L.set()
self.sm.add_widget(self.L) return self.sm

class HotelMangementSystemApp(App): def build(self):
inspector.create_inspector(Window,Main) X = Main()
return X.Start()

if __name__ == '__main__':
HotelMangementSystemApp().run()

```

SignUp.py

```

from sys import path
path.append('/usr/lib/python3/dist-packages/')
from kivy.app import App
from kivy.lang import Builder
from Screen.Login import Title
from kivy.uix.screenmanager import ScreenManager,Screen
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput

```

```

from kivy.modules import inspector
from kivy.core.window import Window
from kivy.uix.togglebutton import ToggleButton
from kivy.uix.button import Button
from kivy.uix.popup import Popup
import mysql.connector as db
hmdb =
db.connect(host="localhost",user="root",passwd="1997",db="hmsdb")
cursor = hmdb.cursor()
Builder.load_string("""
<STitle>:

```

```

orientation: 'vertical'
size_hint : 1, .08
pos_hint : {'top' : .9,'center_x' : 0.5}
canvas:

```

```

Color :
rgba : 1,.8353,.3098,1
Rectangle:

```

```

size : self.size
pos : self.pos
Label:
text : "Sign Up"
color : 0,0,0,1
bold : True
font_size : 35

```

```

<SignUpBg>:
canvas:

```

```

Color:
rgba : 1,.8353,.3098,1
Rectangle :
size : self.size
pos : self.pos
""")

```

```

class STitle(BoxLayout):
# Sign up title
pass

```

```

class RButton(BoxLayout):
# Radio button for selecting gender
def set(self):

self.selectMale = False
self.selectFemale = False
self.orientation = 'horizontal'
self.spacing = 10
self.padding = (10,10)
self.l = Label(text = 'Gender',size_hint = (1.5,1),color = (0,0,0,1))
self.add_widget(self.l)
self.m = ToggleButton(text = 'Male',group = 'gender',on_press =

self.selectM,background_color=(.1254,.0392,.2117,1))
self.add_widget(self.m)
self.f = ToggleButton(text = 'Female', group = 'gender',on_press =

self.selectF,background_color=(.1254,.0392,.2117,1))
self.add_widget(self.f)
def selectM(self,x):
self.selectMale = True self.selectFemale = False

def selectF(self,x):
self.selectFemale = True self.selectMale = False

class DOB(BoxLayout):
# Frame of widgets to get date of birth
def set(self):

self.orientation = 'horizontal'
self.padding = (5,5)
self.spacing = 10
self.l = Label(text= 'Date Of Birth', color = (0,0,0,1))
self.d = TextInput(hint_text = 'DD',size_hint = (.3,1),write_tab =
False,input_filter = 'int') self.m = TextInput(hint_text = 'MM',size_hint =
(.3,1),write_tab = False,input_filter = 'int') self.yy = TextInput(hint_text
= 'YYYY',size_hint = (.5,1),write_tab = False,input_filter = 'int')
self.add_widget(self.l)

```

```
self.add_widget(self.d)
self.add_widget(self.m)
self.add_widget(self.yy)
```

```
class SignUpBg(BoxLayout): # Background for SignUP page
def set(self):
```

```
self.orientation = 'horizontal' self.size_hint = (1,.82)
self.FL = FirstList()
self.FL.set()
self.add_widget(self.FL) self.SL = SecondList()
self.SL.set()
self.add_widget(self.SL)
```

```
class IL(BoxLayout):
# Frame which group the label and input for various attributes
def set(self,t,ps = False, input_f = None):
```

```
self.t = t
self.add_widget(Label(text = t + ' : ', color = (0,0,0,1)))
self.ti = TextInput(hint_text = t ,password = ps,write_tab = False,
input_filter = input_f ) self.add_widget(self.ti)
class PopUp(Popup):
```

```
def set(self,title,content):
self.title = title
self.content = Label(text = content ) self.size_hint = (None,None)
self.size = (200,200)
```

```
class FirstList(BoxLayout):
# Contais Frames at the left side of screen
def set(self):
```

```
self.orientation = 'vertical'
self.padding = (10,10)
self.spacing = 10
self.FirstName = self.aw('First Name')
self.LastName = self.aw('Last Name')
```



```

self.Email = self.aw('Email')
self.PhoneNo = self.aw('Phone No.',inp='int') self.Username =
self.aw('Username')
self.PassWord = self.aw('Password',ps=True) self.CPassWord =
self.aw('Confirm Password',ps=True) self.SAddress = self.aw('Street
Address',inp = None) self.City = self.aw('City')

```

```

def aw(self,t,ps= False, inp = None): self.w = IL()
self.w.set(t,ps,input_f=inp) self.add_widget(self.w)
return self.w

```

```

def check(self):
if self.CPassWord.ti.text != self.PassWord.ti.text:
self.PE = PopUp()
self.PE.set('Password Differ', 'Your Password do not match with
confirm password') self.PE.open()
return False

```

```

return True

```

```

def getdata(self):
self.FirstListItem = []
if self.check() == True:

```

```

self.FirstListI =
[self.FirstName,self.LastName,self.Email,self.PhoneNo,self.Username,
self.CPassWord,self.SAddress,self.City]

```

```

for i in self.FirstListI:
self.FirstListItem.append(i.ti.text)
return self.FirstListItem
class SButton(BoxLayout):
# Sign up button and back button
def set(self):
self.backtl = False
self.signupT = False
self.orientation = 'horizontal'
self.spacing = 10

```

```
self.b = Button(text = 'Sign Up',on_press =
self.SignUpButton,background_color=(.1254,.0392,.2117,1))
self.add_widget(self.b)
self.back = Button(text = 'Back', on_press =
self.backtologin,background_color=(.1254,.0392,.2117,1))
self.add_widget(self.back)
```

```
def SignUpButton(self,a): self.signupT = True
def backtologin(self,x): self.backtl = True
```

```
class SecondList(BoxLayout):
# Consist of Frames at the right side of the screen def set(self):
```

```
self.padding = (10,10)
self.spacing = 10
self.orientation = 'vertical'
self.PinCode = self.aw('Pincode')
self.State = self.aw('State')
self.Country = self.aw('Country')
self.Aadhaar = self.aw('Aadhaar No.')
self.Licence = self.aw('Licence No.')
self.Other = self.aw('Other Id No.')
self.g = RButton()
self.g.set()
self.add_widget(self.g)
self.dob = DOB()
self.dob.set()
self.add_widget(self.dob)
self.su = SButton()
self.su.set()
self.add_widget(self.su)
def aw(self,t):
```

```
w = IL()
w.set(t)
self.add_widget(w) return w
```

```

def check(self):
self.PinCode = int(self.PinCode)
def getdata(self):

self.SecondListl = [self.PinCode, self.State, self.Country,
self.Aadhaar, self.Licence, self.Other,] self.SecondListItem = []
for i in self.SecondListl:

self.SecondListItem.append(i.ti.text)
if self.g.selectMale == True :
self.SecondListItem.append('Male')
elif self.g.selectFemale == True:
self.SecondListItem.append('Female')
self.d = self.dob.d.text
self.m = self.dob.m.text
self.y = self.dob.yy.text
self.SecondListItem.append((str(int(self.d))+ '/' +str(int(self.m))+ '/' +str(int(self.y)))) return self.SecondListItem

class SignUpScreen(Screen,BoxLayout): # main screen for sign up
def set(self):

self.name = 'signup'
self.orientation = 'vertical'
self.T = Title()
self.T.set()
self.add_widget(self.T,index = 0) self.add_widget(STitle(),index = 0)
self.X = SignUpBg()
self.X.set()
self.X.SL.su.b.on_press = self.getdata self.add_widget(self.X,index =
1) def getdata(self):

self.X.FL.getdata()
self.fl = self.X.FL.FirstListItem
self.X.SL.getdata()
self.sl = self.X.SL.SecondListItem
self.total = self.fl + self.sl
self.X.SL.su.backtl = True

```

```
self.query = 'INSERT INTO userdetails VALUES ('  
for i in range(len(self.total)):
```

```
self.query += ""'+self.total[i]+'""
```

```
if i == len(self.total)-1:
```

```
self.query = self.query
```

```
else:
```

```
self.query += ','
```

```
self.query += ')'
```

```
cursor.execute(self.query)
```

```
self.q = "insert into logindetails values
```

```
('{'','}','Customer')".format(self.total[4],self.total[5])
```

```
cursor.execute(self.q)
```

```
hmdb.commit()
```

```
class SignUpApp(App):
```

```
# Create local app
```

```
def build(self):
```

```
self.sm = ScreenManager()
```

```
self.a = SignUpScreen()
```

```
self.a.set()
```

```
self.sm.add_widget(self.a)
```

```
inspector.create_inspector(Window, self.a) return self.sm
```

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

Section.py

```
from sys import path
```

```
path.append('/usr/lib/python3/dist-packages/')
```

```
from kivy.app import App
```

```
from kivy.lang import Builder
```

```
from kivy.uix.screenmanager import ScreenManager,Screen from
```

```
Screen.Login import Title
```

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

```
from kivy.uix.image import Image
```

```
from kivy.uix.behaviors import ButtonBehavior
```

```

from kivy.uix.button import Button
RSEL = False
LSEL = False
Builder.load_string("""
<FrameL>:

canvas :

Color :
rgba : 0,0,0,1
Rectangle :
size : self.size
pos : self.pos
""")
class CustomButtonR(ButtonBehavior,Image):
# Custom button to make image act as button to select Resturant def
__init__(self, **kwargs):
super(CustomButtonR, self).__init__(**kwargs)
self.source = 'res.jpg'
self.allow_stretch = True
self.keep_ratio = False
self.RSEL = False

def on_release(self): self.RSEL= True

class CustomButtonL(ButtonBehavior,Image):
#Custom button to make image act as button to select Logging def
__init__(self, **kwargs):

super(CustomButtonL, self).__init__(**kwargs)
self.source = 'log.jpg'
self.allow_stretch = True
self.keep_ratio = False
self.LSEL = False

def on_release(self):
self.LSEL = True
class FrameL (BoxLayout):

```

```

# This is parent frame
def set(self):
    self.orientation = 'vertical'
    self.padding = (10,10)
    self.spacing = 10

class LabelF(Button):
    # This class create label at bottom of image def set(self,text):

    self.text = text
    self.background_normal= "
    self.background_color = (1,.4392,.2627,1) self.color = (0,0,0,1)

class Main(BoxLayout):
    def set(self):
        #This function set all the objects in the class Main and set their
        appropriate values self.orientation = 'horizontal'
        self.R = CustomButtonR(size_hint = (1,.9))
        self.L = CustomButtonL(size_hint = (1,.9))
        self.RB = FrameL()
        self.RB.set()
        self.RL = LabelF(size_hint = (1,.1),on_press = self.ResS )
        self.RL.set('Resturant')
        self.RB.add_widget(self.R)
        self.RB.add_widget(self.RL)
        self.LL = LabelF(size_hint = (1,.1), on_press = self.LogS)
        self.LL.set('Lodging')
        self.LB = FrameL()
        self.LB.set()
        self.LB.add_widget(self.L)
        self.LB.add_widget(self.LL)
        self.add_widget(self.RB)
        self.add_widget(self.LB)

    def ResS(self,s):
        # Fuction to tell that Resturant is selected self.R.RSel = True
    def LogS(self,x):

```

```
# Fuction to tell that Lodging is selected self.L.LSel = True
```

```
class SectionScreen(Screen,BoxLayout):
def set(self):
# Section Screen set the values to appropate postion self.name =
'section'
self.orientation = 'vertical'
self.T = Title()
self.T.set()
self.add_widget(self.T,index = 0)
self.m = Main(size_hint = (1,.9))
self.m.set()
self.add_widget(self.m,index = 0)
```

```
class SectionApp(App):
# SectionApp create application of individual application to run def
build(self):
```

```
self.sm = ScreenManager()
self.s = SectionScreen()
self.s.set()
self.sm.add_widget(self.s)
return self.sm
```

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

Resturant.py

```
from sys import path
path.append('/usr/lib/python3/dist-packages/')
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager,Screen
from kivy.modules import inspector
from kivy.core.window import Window
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.lang import Builder
from kivy.uix.tabbedpanel import TabbedPanel,TabbedPanellItem from
kivy.uix.scrollview import ScrollView
```

```
from kivy.uix.gridlayout import GridLayout
from kivy.uix.stacklayout import StackLayout
from kivy.uix.button import Button
from kivy.clock import Clock
import random
import mysql.connector as db
hmdb =
db.connect(host="localhost",user="root",passwd="1997",db="hmsdb")
cursor = hmdb.cursor()
```

```
Builder.load_string("""
<ResTitle>:
orientation: 'vertical'
size_hint : 1, .1
canvas:
Color :
rgba : .9411,.3843,.5725,1 Rectangle:
size : self.size
pos : self.pos
Label:
text : "Resturant"
color : 0,0,0,1
bold : True
font_size : 50
```

```
<ResMenu>:
canvas:
```

```
Color:
rgba : 0,1,0,1
Rectangle:
size : self.size pos : self.pos
<Orders>:
canvas:
Color:
rgba : 0,1,1,1 Rectangle:
size : self.size pos : self.pos
<Item>:
```



```
canvas:  
Color:  
rgba : 0,1,.5,1 Rectangle:  
size : self.size pos : self.pos
```

```
<Order>:  
canvas:  
Color:  
rgba : 0,0,0,1
```

```
Rectangle:  
size: self.size pos : self.pos
```

```
""")  
cartitem = []  
nud = False
```

```
class Order(BoxLayout):  
def set(self,item):  
self.rm = False
```

```
self.orientation = 'horizontal'  
self.size_hint = (1,None)  
self.height = 44  
self.name = Label(text = item[0],size_hint = (.4,1))  
self.add_widget(self.name)  
self.quantity = Label(text = str(item[2]),size_hint = (.3,1))  
self.add_widget(self.quantity)  
self.price = Label(text = str(int(item[2]) * int(item[1])),size_hint =(.2,1))  
self.add_widget(self.price)  
self.remove = Button(text = "X",size_hint = (.1,1),on_press =  
self.removeOrder) self.add_widget(self.remove)  
return self  
def removeOrder(self,a):  
  
self.rm =True
```

```

class Orders(BoxLayout):
    global cartitem
    global nud
    def set(self):

        self.orientation = 'vertical'
        self.size_hint = (.3,1)
        self.padding = (10,10)
        self.spacing = 10
        self.cart = Label(text='Your Cart',color=(0,0,0,1),size_hint=(1,None),height=15) self.add_widget(self.cart)
        self.bl = BoxLayout(orientation='horizontal',size_hint = (1,None),height=15) self.i = Label(text='Items',color=(0,0,0,1),size_hint= (.4,1))
        self.bl.add_widget(self.i)
        self.q = Label(text='Quantity',color=(0,0,0,1),size_hint= (.3,1))
        self.p = Label(text='Price',color=(0,0,0,1),size_hint= (.3,1))
        self.bl.add_widget(self.q)
        self.bl.add_widget(self.p)
        self.add_widget(self.bl)
        self.gl = GridLayout(cols=1, spacing=10, size_hint_y=None)
        self.gl.bind(minimum_height = self.gl.setter('height'))
        self.sv = ScrollView(size_hint=(1, 1), size=(self.width, self.height),do_scroll_x = False) self.add_widget(self.sv)
        self.tbl = BoxLayout(orientation = 'horizontal',size_hint = (1,None),height = 15) self.total = 0
        self.tbl.add_widget(Label(text='Total',size_hint=(.5,1),color = (0,0,0,1))) self.totalprice = Label(text=str(self.total),size_hint=(.5,1),color = (0,0,0,1)) self.tbl.add_widget(self.totalprice)
        self.add_widget(self.tbl)
        self.sbl = BoxLayout(orientation = 'horizontal',size_hint = (1,None),height = 15)
        self.sbl.add_widget(Button(text='Save',size_hint = (.5,1),on_press = self.back)) self.sbl.add_widget(Button(text = 'Pay Off',size_hint = (.5,1),on_press = self.back)) self.add_widget(self.sbl)
        self.shown = []
        self.obj = []

```

```
self.bk = False  
nud = False
```

```
def back(self,a):  
    self.bk = True  
    self.query = 'insert into reslog values (%s,%s)%'  
    (self.total,random.randint(10,10000)) cursor.execute(self.query)  
    hmdb.commit()
```

```
def up(self,a):  
    global nud  
    global cartitem  
    self.similar = False  
    for i in self.obj:
```

```
        if i.rm == True:  
            self.gl.remove_widget(i)  
            i.rm = False
```

```
    if nud == True:  
        self.sv.remove_widget(self.gl)  
        for i in range(len(cartitem)):
```

```
            for j in range(len(self.shown)): if self.shown[j][0] == cartitem[i][0]:  
                self.similar = True  
                self.objj = j  
                self.obji = i
```

```
    if self.similar == True:  
        self.obj[self.obji].price.text = str(int(cartitem[self.obji][1]) *  
        int(cartitem[self.obji][2])) self.obj[self.obji].quantity.text =  
        str(cartitem[self.obji][2])  
        self.similar = False
```

```
    else:  
        self.obj.append(Order())  
        self.obj[i] = self.obj[i].set(cartitem[i])  
        self.gl.add_widget(self.obj[i])
```

```
self.a = cartitem[i]
self.shown.append(self.a)
```

```
nud = False
self.total = 0
for i in range(len(self.shown)):
```

```
self.total += int(self.obj[i].price.text)
self.totalprice.text = str(self.total)
self.sv.add_widget(self.gl)
```

```
else:
    pass
```

```
class Item(BoxLayout): global cartitem
global nud
```

```
def set(self,row):
self.orientation = 'vertical'
self.row = row
self.size_hint = (.3,.2)
self.name = Label(text=row[0],size_hint=(1,0.5),color = (0,0,0,1))
self.add_widget(self.name)
self.bl = BoxLayout(orientation = 'horizontal', size_hint = (1,.5))
self.price = Label(text='Price :'+str(row[1]),size_hint = (.5,1),color =
(0,0,0,1)) self.bl.add_widget(self.price)
self.addtocart = Button(text='Add to Card', size_hint =
(.5,1),on_press= self.additem) self.bl.add_widget(self.addtocart)
self.add_widget(self.bl)
return self
```

```
def additem(self,a):
global nud
similar = False
nud = True
if cartitem==[]:
```

```

cartitem.append(list(self.row)) else:
for i in range(len(cartitem)): if cartitem[i][0]==self.row[0]: cartitem[i][2]
+=1
similar = True
break
else:
similar = False
if similar == False:
cartitem.append(list(self.row))

```

```

class Items(TabbedPanellItem):
def set(self,text):
cursor.execute('select * from {}'.format(text))
row = cursor.fetchone()
self.sl = StackLayout(orientation = 'lr-tb',padding=(10,10),spacing =
10) self.l = []
i = 0

```

```

while row is not None:
self.l.append(Item())
self.l[i] = self.l[i].set(row) self.sl.add_widget(self.l[i]) i += 1
row = cursor.fetchone()

```

```

self.add_widget(self.sl)

```

```

class ResMenu(TabbedPanel,BoxLayout): def set(self):
self.orientaion = 'horizontal'
self.do_default_tab = False
self.starter = Items(text='Starter')
self.starter.set('starter')
self.add_widget(self.starter)
self.default_tab = self.starter
self.maincourse = Items(text= 'Main Course')
self.maincourse.set('maincourse')
self.add_widget(self.maincourse)
self.breads = Items(text='Breads')
self.breads.set('breads')
self.add_widget(self.breads)

```

```
self.extras = Items(text="Extras")
self.extras.set('extras')
self.add_widget(self.extras)
```

```
class ResBg(BoxLayout):
def set(self):
self.orientation = 'horizontal' self.size_hint = (1,.9)
self.m = ResMenu()
self.m.set()
self.o = Orders()
self.o.set()
self.add_widget(self.o) self.add_widget(self.m) class
ResTitle(BoxLayout):
```

```
# Main title of the Application
def set(self):
self.size_hint = (1, .1)
self.pos_hint = {'top': 1, 'center_x': 0.5}
```

```
class ResScreen(Screen,BoxLayout): def set(self):
self.name = "CustResScreen" self.orientation = 'vertical' self.Ti =
ResTitle()
self.Ti.set()
self.add_widget(self.Ti)
self.rb = ResBg()
self.rb.set()
self.add_widget(self.rb)
```

```
class ResScreenM(ScreenManager): def set(self):
self.R = ResScreen()
self.R.set()
self.add_widget(self.R)
```

```
class ResScreenApp(App):
def build(self):
self.s = ResScreenM()
self.s.set()
```

```
Clock.schedule_interval(self.s.R.rb.o.up, 1.0 / 60.0)
inspector.create_inspector(Window, self.s) return self.s
```

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

Lodging.py

```
from sys import path
path.append('/usr/lib/python3/dist-packages/')
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager,Screen
from Screen.Login import Title
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder
from kivy.uix.stacklayout import StackLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput
from kivy.clock import Clock
import datetime
import random
now = datetime.datetime.now()
import mysql.connector as db
hmdb =
db.connect(host="localhost",user="root",passwd="1997",db="hmsdb")
cursor = hmdb.cursor()
```

```
Builder.load_string("""
```

```
<LodBg>:
```

```
canvas:
```

```
Color:
```

```
rgba : 0,0,0,1
```

```
Rectangle:
```

```
size : self.size
```

```
pos : self.pos
```

<Room>

canvas:

Color:

rgba: .7,.5,.7,1

Rectangle:

size : self.size

pos : self.pos

""

```
class inLabel(BoxLayout):
```

```
def set(self,lt):
```

```
self.orientation = 'horizontal'
```

```
self.size_hint = (1,.5)
```

```
self.label = Label(text =lt + ' : ', size_hint = (.5,1)) self.ti =
```

```
TextInput(hint_text = lt , size_hint = (.5,1))
```

```
class GuestBg(BoxLayout):
```

```
def set(self):
```

```
self.size_hint = (.3,7)
```

```
self.orientation = 'vertical'
```

```
self.spacing = 10
```

```
self.padding = (10,10)
```

```
self.checkinT = False
```

```
self.title = Label(text='Enter Guest Details',size_hint =  
(1,None),height=20) self.add_widget(self.title)
```

```
self.fn = inLabel()
```

```
self.fn.set('First Name')
```

```
self.add_widget(self.fn)
```

```
self.ln = inLabel()
```

```
self.ln.set('Last Name')
```

```
self.add_widget(self.ln)
```

```
self.ids = inLabel()
```

```
self.ids.set('Id No.')
```

```
self.add_widget(self.ids)
```

```
self.phoneno = inLabel()
```

```
self.phoneno.set('Phone No')
```



```
self.checkin = Button(text = 'Check In' , on_press = self.cin)
self.add_widget(self.checkin)
```

```
def cin(self,a):
self.checkinT = True
```

```
class GuestDataScreen(Screen): def set(self,rmno):
self.rmno = rmno
self.T = Title()
self.T.set()
self.add_widget(self.T) self.gr = GuestBg()
self.gr.set()
self.add_widget(self.gr)
```

```
class Room(BoxLayout): def set(self,x):
self.rcin = False self.rcout = False self.cin = False
self.cout = False self.orientation = 'vertical'
self.size_hint = (.3,.2)
self.room = str(x+1)
self.roomno = Label(text = 'Room No. : '+str(x+1),size_hint = (1,.5))
self.add_widget(self.roomno)
self.bl = BoxLayout(size_hint = (1,.5))
self.checkin = Button(text = 'Check In',on_press = self.roomcheckin)
self.bl.add_widget(self.checkin)
self.checkout = Button(text = 'Check Out', on_press =
self.roomcheckout) self.bl.add_widget(self.checkout)
self.status = Label(text = 'Available')
self.bl.add_widget(self.status)
self.add_widget(self.bl)
```

```
def roomcheckin(self,a):
self.rcin = True
self.cin = True
self.status.text = 'UnAvailable'
```

```
def roomcheckout(self,a): self.rcin = False
self.rcout = True
```

```
self.cout = True
self.status.text = "Available"
```

```
class LodBg(StackLayout):
def set(self):
self.size_hint = (1,.9)
self.orientation = 'lr-tb'
self.padding = (10,10)
self.spacing = 10
self.roomsno = 10
self.rooms = []
for i in range(self.roomsno):
```

```
self.rooms.append(Room()) self.rooms[i].set(i)
self.add_widget(self.rooms[i])
```

```
class LodScreen(Screen):
def set(self):
self.name = "LodgingScreen" self.T = Title()
self.T.set()
self.add_widget(self.T, index = 0) self.L = LodBg()
self.L.set()
self.add_widget(self.L,index = 1)
self.bl = StackLayout(orientation = 'rl-tb',size_hint = (1,None),height =
30) self.bl.add_widget(Button(text = 'Back',size_hint= (.1,1),on_press
= self.back)) self.add_widget(self.bl)
self.bk = False
```

```
def update(self,a):
for i in range(10):
if self.L.rooms[i].rcin == True and self.L.rooms[i].cin == True:
```

```
self.d = '{}/{}/{}'.format(now.day, now.month, now.year)
self.t = '{}: {}: {}'.format(now.hour, now.minute, now.second)
self.queryin = 'insert into lodgedetails values ("{}","{}","{}","
{}","in")'.format(self.d,
```

```

self.t,random.randint(10,10000),self.L.rooms[i].room)
cursor.execute(self.queryin)
self.L.rooms[i].cin = False
hmdb.commit()

elif self.L.rooms[i].rcout == True and self.L.rooms[i].cout == True:
self.d = '{}/{}/{}'.format(now.day, now.month, now.year)
self.t = '{}: {}: {}'.format(now.hour, now.minute, now.second)
self.queryout = 'insert into lodgedetails values ("{}","{}","{}","{}",
"out")'.format(self.d,

self.t,random.randint(10,10000),self.L.rooms[i].room)
self.L.rooms[i].cout = False
cursor.execute(self.queryout)
hmdb.commit()

def back(self,a): self.bk = True

class LodScreenApp(App):
def build(self):
self.L = LodScreen()
self.L.set()
self.sm = ScreenManager()
self.sm.add_widget(self.L)
Clock.schedule_interval(self.L.update, 1.0/30.0) return self.L

if __name__ == '__main__':
LodScreenApp().run()
__init__.py print('This is Screen folder')

```

Section 7.3: Chat Server App

This chat application can allows to people to chat. The purpose of this code is for you to read and understand the code and try to use it to create an application for yourself.

main.py

```
import kivy
import socket
import threading
import time
import os
from kivy.animation import Animation
from kivy.uix.boxlayout import BoxLayout from
kivy.uix.screenmanager import *
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout from kivy.uix.gridlayout
import GridLayout from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.button import Button
from kivy.uix.scrollview import ScrollView from kivy.uix.carousel
import Carousel
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput from kivy.uix.filechooser
import *
from kivy.properties import *
from kivy.clock import Clock
from kivy.core.audio import SoundLoader
```

```
##### global WlApp
from ServerSocket import *
from ClientSocket import *
from ClientInformation import *
from Task import *
from Chatroom import *
from ChatroomCollector import *
from FileObject import *
from Invitation import *
from kivy.core.window import Window Window.size = (350, 600)
```

```
class StartupScreen(Screen):
    def __init__(self, **kwargs):
        super(StartupScreen, self).__init__(**kwargs) self.popup = None
```

```
def openHostPopup(self):
    notification = BoxLayout(orientation="vertical")
    label = Label(text="Please turn on Hotspot\n before start hosting",
    font_size="20sp") closeButton = Button(text="Close",
    background_color=(0.000, 0.361, 0.659, 1)) startHostButton =
    Button(text="Start Hosting", background_color=(0.000, 0.361, 0.659,
    1))
```

```
closeButton.size_hint = 1, .3 startHostButton.size_hint = 1, .3
```

```
notification.add_widget(label)
notification.add_widget(startHostButton)
notification.add_widget(closeButton)
```

```
self.popup = Popup(title='CAUTION' ,content=notification, size_hint=
(.8, .5), auto_dismiss=True,
background='pictures/backgroundPopup.png')
self.popup.open()
closeButton.bind(on_press=self.popup.dismiss)
startHostButton.bind(on_press=self.startHosting)
```

```
def getInputStartup(self):
    WIAApp.username = self.nameInput.text.upper()
    WIAApp.status = ""
    WIAApp.ip = self.ipInput.text
    WIAApp.port = int(self.portInput.text)
```

```
def login(self):
    try:
        self.getInputStartup()
        self.startClient(WIAApp.ip, WIAApp.port, WIAApp.username)
        WIAApp.current = "MainUIScreen"
        WIAApp.mainUIScreen.profileArea.idButton.text =
        WIAApp.clientInfo.getID()
        WIAApp.mainUIScreen.profileArea.nameButton.text =
        WIAApp.username WIAApp.mainUIScreen.receiveData_thread.start()
```

```
except OSError:
popup = Popup(title="ERROR!", content=Label(text="There is no any
host in server.", font_size="18sp"),
auto_dismiss=True, size_hint=(0.8, 0.2),
background='pictures/backgroundPopup.png', title_size='20sp')
popup.open()
```

```
def startHosting(self, instance):
self.getInputStartup()
WlApp.serverSocket = ServerSocket(WlApp.ip, WlApp.port)
WlApp.serverSocket.start()
self.popup.dismiss()
self.login()
```

```
def startClient(self, ip, port, username):
WlApp.clientSocket = ClientSocket(ip, int(port), username) # Start
client socket loop WlApp.clientSocket.connect()
```

```
# Send the client information to the server
WlApp.clientInfo = ClientInformation(username, WlApp.status,
WlApp.clientSocket.getAddr(),
```

```
None)
initialTask = Task("Submit ClientInfo", WlApp.clientInfo)
WlApp.clientSocket.sendTask(initialTask)
```

```
class InAppNotification(BoxLayout):
def __init__(self, **kwargs):
super(BoxLayout, self).__init__(**kwargs)
```

```
class MainUIScreen(Screen):
def __init__(self, **kwargs):
super(MainUIScreen, self).__init__(**kwargs)
self.listofScreen = ["contact", "history", "request"]
self.receiveData_thread = threading.Thread(target=self.receiveData)
self.filename = None
self.filePath = None
self.fileSize = None
```

```
self.messageSound = True
self.inviteSound = True
self.sound_msg = SoundLoader.load('sounds/messageSound.mp3')
self.sound_invite = SoundLoader.load('sounds/inviteSound.mp3')
```

```
def openStatusInput(self):
    container = BoxLayout(orientation='vertical')
    self.statusInput = TextInput(multiline=False)
    confirmButton = Button(text='Confirm', background_color=(0.000,
0.361, 0.659, 1)) container.add_widget(self.statusInput)
    container.add_widget(confirmButton)
    popup = Popup(title='Input your status', content=container,
title_size='20sp',
```

```
auto_dismiss=True, size_hint=(.7, .2),
background='pictures/backgroundPopup.png') popup.open()
confirmButton.bind(on_press=self.changeStatus)
def changeStatus(self, instance):
```

```
    status = self.statusInput.text[:20]
    self.profileArea.statusButton.text = status
    WlApp.clientInfo.setStatus(status)
    task = Task("Change Status", WlApp.clientInfo)
    WlApp.clientSocket.sendTask(task)
```

```
def updateContactStatus(self, task, contactContainer): clientInfo =
task.getData()
id = clientInfo.getID()
name = clientInfo.getName()
newStatus = clientInfo.getStatus()
```

```
for child in contactContainer.children:
    if child.idButton.text == id and child.nameButton.text == name:
        child.statusButton.text = newStatus
```

```
def changeScreen(self, name):
    target_idx = self.listofScreen.index(name)
    self.screenSlider.load_slide(self.screenSlider.slides[target_idx])
```

```
def showNotification(self, title, detail): self.inAppNotification.title.text =  
title self.inAppNotification.detail.text = detail
```

```
anim = Animation(pos=(0, self.height-self.inAppNotification.height),  
duration=0.25) anim.start(self.inAppNotification)  
Clock.schedule_once(self.hideNotification, 3)
```

```
def hideNotification(self, data):  
anim = Animation(pos=(-self.width, self.height -  
self.inAppNotification.height), duration=0.25)  
anim.start(self.inAppNotification)
```

```
def moveto_chatroom(self, id, name):  
WlApp.transition = SlideTransition(direction="up") WlApp.current =  
"ChatroomScreen"  
self.setClientTarget(id, name)
```

```
def moveto_groupchat(self, gname, creatorID): WlApp.transition =  
SlideTransition(direction="up") WlApp.current = "GroupChatScreen"  
WlApp.groupchatScreen.roomName.text = gname  
WlApp.groupchatScreen.roomCreatorID = creatorID  
WlApp.currentChatroom =  
WlApp.chatroomCollector.getRoomByRoomName(gname, creatorID)  
WlApp.groupchatScreen.loadDataGroupChatroom(WlApp.currentCh  
atroom)
```

```
def moveto_createGroup(self):  
WlApp.transition = SlideTransition(direction="up") WlApp.current =  
"CreateGroupScreen"  
WlApp.createGroupScreen.listAllContact()
```

```
def isNewHisComp(self, container, obj):  
for element in container:  
if isinstance(element, HistoryComponent) and isinstance(obj,  
HistoryComponent): if element.idButton.text == obj.idButton.text:  
return False  
elif isinstance(element, HistoryGroupComponent) and isinstance(obj,
```



```
HistoryGroupComponent):  
if ( element.gnameButton.text == obj.gnameButton.text and  
element.creatorID.text == obj.creatorID.text):  
return False
```

```
return True
```

```
def sortContactByName(self):  
contactScrollView =  
self.screenSlider.contactScreen.contactScrollView for i in  
range(len(contactScrollView.children) - 1):  
  
for j in range(len(contactScrollView.children) - 1):  
if contactScrollView.children[j].nameButton.text <  
contactScrollView.children[j +  
  
1].nameButton.text:  
temp = contactScrollView.children[j]  
contactScrollView.children[j] = contactScrollView.children[j + 1]  
contactScrollView.children[j+1] = temp
```

```
# Move the key to front from the list  
def move_to_front(self, key, mylist):  
mylist.remove(key)  
mylist.insert(len(mylist), key)
```

```
# When send a new message  
def updateHistoryType_1(self, msg):  
historyScrollView = self.screenSlider.historyScreen.historyScrollView  
if msg.getGroupName() == None:  
historyComp = HistoryComponent(WIApp.clientTargetID,  
WIApp.clientTargetName, msg.getText())
```

```
elif msg.getGroupName() != None:  
historyComp = HistoryGroupComponent(msg.getGroupName(),  
msg.getRoomCreatorID(),  
msg.getText())
```

```

# Add a new History component to its scroll view
if self.isNewHisComp(historyScrollView.children, historyComp) ==
True: historyScrollView.add_widget(historyComp,
len(historyScrollView.children)) # If it already exist, rotate the lastest
to the front
elif self.isNewHisComp(historyScrollView.children, historyComp) ==
False: for i in range(len(historyScrollView.children)):

if isinstance(historyScrollView.children[i], HistoryComponent):
if ( historyScrollView.children[i].idButton.text ==
msg.getReceiverAddr()[0] and msg.getGroupName() == None ):
historyScrollView.children[i].lastestMsg.text = msg.getText()
self.move_to_front(historyScrollView.children[i],
historyScrollView.children)

elif isinstance(historyScrollView.children[i], HistoryGroupComponent):
if ( historyScrollView.children[i].gnameButton.text ==
msg.getGroupName() and historyScrollView.children[i].creatorID.text
== msg.getRoomCreatorID()):
historyScrollView.children[i].lastestMsg.text = msg.getText()
self.move_to_front(historyScrollView.children[i],
historyScrollView.children)

# When got an incoming message
def updateHistoryType_2(self, msg):
historyScrollView = self.screenSlider.historyScreen.historyScrollView
if msg.getGroupName() == None:
historyComp = HistoryComponent(msg.getOwnerId(),
msg.getOwnerName(), msg.getText())
if msg.getGroupName() != None:
historyComp = HistoryGroupComponent(msg.getGroupName(),
msg.getRoomCreatorID(), msg.getText())
# Add a new History component to its scroll view
if self.isNewHisComp(historyScrollView.children, historyComp) ==
True:
historyScrollView.add_widget(historyComp,
len(historyScrollView.children)) elif
self.isNewHisComp(historyScrollView.children, historyComp) ==

```

False:

```
for i in range(len(historyScrollView.children)):
```

```
if isinstance(historyScrollView.children[i], HistoryComponent):
```

```
if historyScrollView.children[i].idButton.text == msg.getOwnerID() and  
msg.getGroupName() == None:
```

```
historyScrollView.children[i].lastestMsg.text = msg.getText()
```

```
self.move_to_front(historyScrollView.children[i],
```

```
historyScrollView.children) elif isinstance(historyScrollView.children[i],  
HistoryGroupComponent):
```

```
if ( historyScrollView.children[i].gnameButton.text ==
```

```
msg.getGroupName() and historyScrollView.children[i].creatorID.text  
== msg.getRoomCreatorID()):
```

```
historyScrollView.children[i].lastestMsg.text = msg.getText()
```

```
self.move_to_front(historyScrollView.children[i],
```

```
historyScrollView.children)
```

```
def sendGroupInformation(self, groupChat): task = Task("Update  
Group Members", groupChat) WlApp.clientSocket.sendTask(task)
```

```
def updateGroupMembers(self, task): groupObj = task.getData()
```

```
gname = groupObj.getRoomName() creatorID =
```

```
groupObj.getCreatorID()
```

```
currentGroup =
```

```
WlApp.chatroomCollector.getRoomByRoomName(gname, creatorID)
```

```
gmemberID = groupObj.getMemberIDList() currentGroupID =
```

```
currentGroup.getMemberIDList()
```

```
difference = set(gmemberID).difference(set(currentGroupID)) while  
len(difference) != 0:
```

```
currentGroup.addMemberID(difference.pop())
```

```
def receiveData(self):
```

```
contactScrollView =
```

```
self.screenSlider.contactScreen.contactScrollView inviteScrollView =
```

```
WlApp.createGroupScreen.contactContainer
```

```

while True:
    data = WIAApp.clientSocket.getDataIncome()
    if data != None:

        try:
            task = pickle.loads(data)
            if task.getName() == "New Client":

                self.appendNewContact(task, contactScrollView, inviteScrollView)
            if task.getName() == "Remove Client":
                self.removeContact(task, contactScrollView, inviteScrollView)
            if task.getName() == "Change Status":
                self.updateContactStatus(task, contactScrollView)

            if task.getName() == "Invite to group":
                inviteObj = task.getData()
                title = "Invitation"
                ownerName = inviteObj.getOwnerInfo().getName()
                detail = ownerName + " invite to group: " + inviteObj.getGroupName()
                detail = detail[:20]
                requestContainer = self.screenSlider.requestScreen.requestContainer
                requestContainer.add_widget(RequestComponent(inviteObj.getOwnerInfo().getID(),

                    inviteObj.getOwnerInfo().getName(),
                    inviteObj.getGroupName()))
                if WIAApp.current == "MainUIScreen":
                    self.showNotification(title, detail)
                if WIAApp.current == "ChatroomScreen":
                    WIAApp.chatroomScreen.showNotificationChatroom(title, detail)
                if WIAApp.current == "GroupChatScreen":
                    WIAApp.groupchatScreen.showNotificationChatroom(title, detail)
                if self.inviteSound == True: self.sound_invite.play()

            if task.getName() == "Response Invitation":
                inviteObj = task.getData()
                if inviteObj.getResponse() == "accept":

```

```

gname = inviteObj.getGroupName()
ownerID = inviteObj.getOwnerInfo().getID()
groupChatRoom =
WlApp.chatroomCollector.getRoomByRoomName(gname,

WlApp.clientInfo.getID())
groupChatRoom.addMemberID(ownerID)
self.sendGroupInformation(groupChatRoom)

if task.getName() == "Message":
WlApp.chatroomScreen.updateMessage(task)
msgObj = task.getData()
ownerName = msgObj.getOwnerName()
detail = msgObj.getText()
detail = detail[:20]

if WlApp.current == "MainUIScreen":
self.showNotification(ownerName, detail)
if WlApp.current == "ChatroomScreen":
WlApp.chatroomScreen.showNotificationChatroom(ownerName,
detail)
if WlApp.current == "GroupChatScreen":
WlApp.groupchatScreen.showNotificationChatroom(ownerName,
detail)
if self.messageSound == True: self.sound_msg.play()
if task.getName() == "Group Message":
WlApp.groupchatScreen.updateGroupMessage(task)

msgObj = task.getData()
gName = msgObj.getGroupName() detail = msgObj.getText()
detail = detail[:20]

if WlApp.current == "MainUIScreen": self.showNotification(gName,
detail)
if WlApp.current == "ChatroomScreen":
WlApp.chatroomScreen.showNotificationChatroom(gName, detail)
if WlApp.current == "GroupChatScreen":
WlApp.groupchatScreen.showNotificationChatroom(gName, detail)

```

```

if self.messageSound == True: self.sound_msg.play()
if task.getName() == "Update Group Members":
self.updateGroupMembers(task)

if task.getName() == "FileDetail":
fileObj = task.getData()
self.filename = fileObj.getFilename() self.fileSize =
fileObj.getFileSize()
self.filePath = "clientFiles/" + self.filename

self.requestFileData
WlApp.clientSocket.clearData() except OverflowError:
pass

except ValueError:
pass
except KeyError:
pass
except EOFError:
pass
except pickle.UnpicklingError: pass
except pickle.PicklingError: pass
except pickle.PickleError: pass

time.sleep(0.1)
def removeContact(self, task, container1, container2): id =
task.getData()
# Remove ContactComponent for x in container1.children: if
x.idButton.text == str(id): container1.remove_widget(x)
# Remove InviteComponent
for y in container2.children:
if y.idButton.text == str(id):
print("remove Invite")
container2.remove_widget(y)

def appendNewContact(self, task, container1, container2):
WlApp.clientInfoList = task.getData()
container1.clear_widgets()

```

```

idx = 0
for x in WIAApp.clientInfoList:
    if x.getID() != WIAApp.clientInfo.getID():

        c = ContactComponent(x.getID(), x.getName())
        container1.add_widget(c, idx)
        idx += 1
        idx = 0
        for x in WIAApp.clientInfoList:

            if x.getID() != WIAApp.clientInfo.getID(): c = InviteComponent()
            c.idButton.text = str(x.getID()) c.nameButton.text = x.getName()
            container2.add_widget(c, idx) idx += 1

def searchAddrByName(self, targetName): for client in
WIAApp.clientInfoList:
    if client.getName() == targetName:

        return client.getAddress()
def setClientTarget(self, targetID, targetName):
WIAApp.clientTargetName = targetName WIAApp.clientTargetID =
targetID

WIAApp.chatroomScreen.roomName.text = WIAApp.username + " and "
+ targetName currentRoom =
WIAApp.chatroomCollector.getRoomByMemberID([WIAApp.clientInfo.ge
tID(), targetID])

if currentRoom not in WIAApp.chatroomCollector.getChatroomList():
    roomName = WIAApp.username + " and " + targetName
    newChatroom = Chatroom(roomName, rType="single")
    newChatroom.addMemberID(WIAApp.clientInfo.getID())
    newChatroom.addMemberID(targetID)
    WIAApp.chatroomCollector.addNewChatroom(newChatroom)

# Choose the current chatroom to load and save the history chat

WIAApp.currentChatroom =
WIAApp.chatroomCollector.getRoomByMemberID([WIAApp.clientInfo.ge

```

```
getID(), targetID]) ## Load private chat
```

```
WlApp.chatroomScreen.loadDataChatroom(WlApp.currentChatroom)
```

```
class InviteComponent(BoxLayout):
```

```
def __init__(self, **kwargs):
```

```
super(BoxLayout, self).__init__(**kwargs) def isSelected(self):
```

```
return self.selection.active
```

```
class RequestComponent(BoxLayout):
```

```
def __init__(self, id, name, groupname, **kwargs):
```

```
BoxLayout.__init__(self, **kwargs)
```

```
self.idButton.text = str(id)
```

```
self.nameButton.text = name
```

```
self.groupButton.text = groupname
```

```
class CreateGroupScreen(Screen):
```

```
def __init__(self, **kwargs):
```

```
super(CreateGroupScreen, self).__init__(**kwargs)
```

```
def listAllContact(self):
```

```
self.contactContainer.clear_widgets() idx = 0
```

```
for client in WlApp.clientInfoList:
```

```
if client.getID() != WlApp.clientInfo.getID(): ic = InviteComponent()
```

```
ic.idButton.text = str(client.getID()) ic.nameButton.text =
```

```
client.getName() self.contactContainer.add_widget(ic, idx) idx += 1
```

```
def removeContact(self): pass
```

```
def moveto_mainUI(self):
```

```
WlApp.transition = SlideTransition(direction="right") WlApp.current =
```

```
"MainUIScreen"
```

```
def sendInvitation(self):
```

```
gname = self.groupNameInput.text creatorID =
```

```
WlApp.clientInfo.getID()
```

```
if len(gname) == 0 or not gname.isalpha():
```

```
popup = Popup(title='Invalid Group Name',
```



```

content=Label(text="Group name must be alphabet!"),

auto_dismiss=True, size_hint=(.7,.2),
background='pictures/backgroundPopup.png') popup.open()
return None

if len(self.contactContainer.children) == 0:
    popup = Popup(title='ERROR', content=Label(text="There is no client
in server!"), auto_dismiss=True, size_hint=(.7, .2),
background='pictures/backgroundPopup.png') popup.open()
return None
if WlApp.chatroomCollector.isExist(gname, creatorID):
    popup = Popup(title='Invalid Group Name', content=Label(text="This
name already exist!"), auto_dismiss=True, size_hint=(.7, .2),
background='pictures/backgroundPopup.png')

popup.open()
return None
receivedAddrs = []
for invc in self.contactContainer.children:

if invc.isSelected() == True:
    receivedAddrs.append(invc.idButton.text)
    inviteObj = Invitation(receivedAddrs, WlApp.clientInfo,
self.groupNameInput.text) task = Task("Invite to group", inviteObj)

# Create widget #
gname = self.groupNameInput.text
group = Chatroom(gname, WlApp.clientInfo.getID(), rType="group")
group.addMemberID(inviteObj.getOwnerInfo().getID()) ## Attach id of
sender WlApp.chatroomCollector.addNewChatroom(group)

groupWidget = GroupChatComponent(gname,
inviteObj.getOwnerInfo().getID(),
inviteObj.getOwnerInfo().getName())
groupContainer =
WlApp.mainUIScreen.screenSlider.contactScreen.groupContainer

```

```
groupContainer.add_widget(groupWidget)
WlApp.clientSocket.sendTask(task)
```

```
class GroupChatComponent(BoxLayout):
    def __init__(self, gname, creatorID, creatorName, **kwargs):
        BoxLayout.__init__(self)
        self.gnameButton.text = gname
        self.creatorID.text = creatorID
        self.creatorName.text = "Creator: " + creatorName
```

```
class HistoryGroupComponent(BoxLayout):
    def __init__(self, gname, creatorID, latestMsg, **kwargs):
        super(BoxLayout, self).__init__(**kwargs)
        self.gnameButton.text = gname
        self.creatorID.text = creatorID
        self.latestMsg.text = latestMsg
```

```
class IDContainer(BoxLayout):
    def __init__(self, **kwargs):
        super(BoxLayout, self).__init__(**kwargs)
```

```
class GroupChatScreen(Screen):
    def __init__(self, **kwargs):
        super(GroupChatScreen, self).__init__(**kwargs)
        self.filePath = None
        self.roomCreatorID = None
```

```
def backup(self):
    currentRoom = WlApp.currentChatroom
    messages = currentRoom.getMsgCollector()
    filename = currentRoom.getRoomName() + '.txt'
    file = open('backupChat/' + filename, 'w')
```

```
for msg in messages:
    print(msg.getOwnerName())
    if msg.getOwnerId() != WlApp.clientInfo.getID():
```

```
        data = msg.getOwnerName() + " @ " + msg.getTimeCreated() + " : "
        + msg.getText() + "\n"
        file.write(data)
```

```
elif msg.getOwnerID() == WIAApp.clientInfo.getID():
data = "You @ " + msg.getTimeCreated() + " : " + msg.getText() + "\n"
file.write(data)
```

```
file.close()
print("Close file")
def showNotificationChatroom(self, title, detail):
if title == WIAApp.clientTargetName or title ==
WIAApp.groupchatScreen.roomName.text: return None
self.inRoomNotification.title.text = title
self.inRoomNotification.detail.text = detail
```

```
anim = Animation(pos=(0, self.height -
self.inRoomNotification.height), duration=0.25)
anim.start(self.inRoomNotification)
Clock.schedule_once(self.hideNotification, 3)
```

```
def listMembers(self):
idContainer = IDContainer()
currentRoom =
WIAApp.chatroomCollector.getRoomByRoomName(self.roomName.te
xt,
```

```
self.roomCreatorID)
idList = currentRoom.getMemberIDList()
for id in idList:
idContainer.container.add_widget(Label(text= "ID: " + str(id),
size_hint_y=None, font_size="18sp"))
popup = Popup(title="All members:", content=idContainer,
auto_dismiss=True, size_hint=(0.7,0.7),
background='pictures/backgroundPopup.png', title_size='20sp')
```

```
popup.open()
def hideNotification(self, data):
anim = Animation(pos=(-self.width, self.height -
self.inRoomNotification.height), duration=0.25)
anim.start(self.inRoomNotification)
```

```

def sendMessageTask(self, groupname, creatorID):
    if self.messageInput.text == "":
        return None
    groupChat =
    WIAApp.chatroomCollector.getRoomByRoomName(groupname,
    creatorID) receiverAddrs = groupChat.getMemberIDList()

    createTime = str(datetime.now())
    createTime = createTime[0:len(createTime) - 7]
    msg = Message(self.messageInput.text, receiverAddrs,

    (WIAApp.username, WIAApp.clientInfo.getID()), groupname, creatorID,
    timeCreated=createTime)

    task = Task("Group Message", msg)
    WIAApp.clientSocket.sendTask(task) WIAApp.currentChatroom =
    groupChat WIAApp.currentChatroom.addMessage(msg)

    if self.messageInput.text != "":
        messageBox = MessageBoxOwner("YOU", msg.getText(),
        msg.getTimeCreated())
        self.groupchatContainer.add_widget(messageBox)

    self.messageInput.text = "" # Clear Message Input
    WIAApp.mainUIScreen.updateHistoryType_1(msg) # Update history
    scroll view when send a new message

def loadDataGroupChatroom(self, room):
    self.messageList = room.getMsgCollector()
    self.groupchatContainer.clear_widgets()
    self.messageInput.bind(on_text_validate=self.on_enter)

    for msg in self.messageList:
        if msg.getGroupName() == room.getRoomName():
            if msg.getOwnerId() != WIAApp.clientInfo.getID():
                messageBox = MessageBoxPartner(msg.getOwnerName(),
                msg.getText(), msg.getTimeCreated())
            elif msg.getOwnerId() == WIAApp.clientInfo.getID():

```

```

messageBox = MessageBoxOwner("YOU", msg.getText(),
msg.getTimeCreated())

self.groupchatContainer.add_widget(messageBox) def
showMenu(self):
anim = Animation(pos=(0, 0), duration=.5) anim.start(self.menu)

def hideMenu(self):
anim = Animation(pos=(-self.width, 0), duration=.5)
anim.start(self.menu)

def updateGroupMessage(self, task): msg = task.getData()
gname = msg.getGroupName() creatorID = msg.getRoomCreatorID()

if WIAApp.currentChatroom == None:
WIAApp.currentChatroom =
WIAApp.chatroomCollector.getRoomByRoomName(gname, creatorID)
# Got the message while talking to another chatroom for room in
WIAApp.chatroomCollector.getChatroomList(): if
room.getRoomName() == gname:
room.addMessage(msg)
## If got any message while talking in the selected chatroom
if room.getRoomName() ==
WIAApp.currentChatroom.getRoomName(): if msg.getOwnerId() !=
WIAApp.clientInfo.getID():
messageBox = MessageBoxPartner(msg.getOwnerName(),
msg.getText(), msg.getTimeCreated())
elif msg.getOwnerId() == WIAApp.clientInfo.getID():
messageBox = MessageBoxOwner("YOU", msg.getText(),
msg.getTimeCreated())
self.groupchatContainer.add_widget(messageBox)
WIAApp.mainUIScreen.updateHistoryType_2(msg) # Group def
openChooserDialog(self):
cd = FileChooserDialog()
popup = Popup(title="Choose a file to send", content=cd, size_hint=
(.8, .8), auto_dismiss=True)
popup.open()

```

```

def selectFile(self, path, filename):
    file = open(os.path.join(path, filename[0]), 'rb')
    fname = os.path.join(path, filename[0])
    fsize = os.path.getsize(fname)
    fname = fname.split("\\")[-1] # Use the last index as a filename

    obj = FileObject(fname, fsize, WIAApp.clientInfo.getID(),
[WIAApp.clientTargetID]) task = Task("Send File", obj)
    WIAApp.clientSocket.sendTask(task)

    data = file.read(1024)
    while data:
        print(">>Sending a file<<")
        WIAApp.clientSocket.sendData(data)
        data = file.read(1024)

    print("Completed sending file!") file.close()

class ChatroomScreen(Screen):
    def __init__(self, **kwargs):
        super(ChatroomScreen, self).__init__(**kwargs) self.messageList =
        None
        self.filePath = None

    def on_enter(self, *args): self.hideMenu()

    def backup(self):
        currentRoom = WIAApp.currentChatroom
        messages = currentRoom.getMsgCollector()
        filename = currentRoom.getRoomName() + '.txt'
        file = open('backupChat/' + filename, 'w')

        for msg in messages:
            print(msg.getOwnerName())
            if msg.getOwnerId() != WIAApp.clientInfo.getID():

                data = msg.getOwnerName() + " @ " + msg.getTimeCreated() + " : "
                + msg.getText() + "\n" file.write(data)

```

```
elif msg.getOwnerId() == WIAApp.clientInfo.getID():
data = "You @ " + msg.getTimeCreated() + " : " + msg.getText() + "\n"
file.write(data)
```

```
file.close()
print("Close file")
def showNotificationChatroom(self, title, detail):
if
title == WIAApp.clientTargetName:
return None
self.inRoomNotification.title.text = title
self.inRoomNotification.detail.text = detail
anim = Animation(pos=(0, self.height -
self.inRoomNotification.height), duration=0.25)
anim.start(self.inRoomNotification)
Clock.schedule_once(self.hideNotification, 3)
```

```
def hideNotification(self, data):
anim = Animation(pos=(-self.width, self.height -
self.inRoomNotification.height), duration=0.25)
anim.start(self.inRoomNotification)
```

```
def moveto_mainUI(self):
WIAApp.transition = SlideTransition(direction="right")
WIAApp.current = "MainUIScreen"
self.hideMenu()
```

```
def loadDataChatroom(self, room):
self.messageList = room.getMsgCollector()
self.chatContainer.clear_widgets()
self.messageInput.bind(on_text_validate=self.on_enter)
```

```
for msg in self.messageList:
if msg.getOwnerId() != WIAApp.clientInfo.getID():
messageBox = MessageBoxPartner(msg.getOwnerName(),
msg.getText(), msg.getTimeCreated())
elif msg.getOwnerId() == WIAApp.clientInfo.getID():
messageBox = MessageBoxOwner("YOU", msg.getText(),
msg.getTimeCreated())
self.chatContainer.add_widget(messageBox)
```

```

def showMenu(self):
    anim = Animation(pos=(0, 0), duration=.5) anim.start(self.menu)

def hideMenu(self):
    anim = Animation(pos=(-self.width, 0), duration=.5)
    anim.start(self.menu)

def sendMessageTask(self):
    if self.messageInput.text == "":
        return None

    createTime = str(datetime.now())
    createTime = createTime[0:len(createTime)-7]
    currentRoom =
    WlApp.chatroomCollector.getRoomByMemberID([WlApp.clientTargetI
    D,

    WlApp.clientInfo.getID()])
    WlApp.currentChatroom = currentRoom

    msg = Message(self.messageInput.text, [WlApp.clientTargetID,
    WlApp.clientInfo.getID()], (WlApp.username,
    WlApp.clientInfo.getID()), timeCreated=createTime )
    task = Task("Message", msg)
    WlApp.clientSocket.sendTask(task)
    WlApp.currentChatroom.addMessage(msg)

    if self.messageInput.text != "":
        messageBox = MessageBoxOwner("YOU", msg.getText(),
        msg.getTimeCreated()) self.chatContainer.add_widget(messageBox)

    WlApp.mainUIScreen.updateHistoryType_1(msg) # Update history
    scroll view when send a new message
    self.messageInput.text = "" # Clear Message Input

def updateMessage(self, task):
    msg = task.getData()
    targetName = msg.getOwnerName() targetID = msg.getOwnerID()
    WlApp.clientTargetID = targetID

```



```

currentRoom =
WIAApp.chatroomCollector.getRoomByMemberID([WIAApp.clientInfo.ge
tID(), targetID])
if currentRoom not in WIAApp.chatroomCollector.getChatroomList():
# Create a new room
roomName = WIAApp.username + " and " + targetName
newChatroom = Chatroom(roomName, rType="single")

newChatroom.addMemberID(WIAApp.clientInfo.getID())
newChatroom.addMemberID(targetID)
WIAApp.chatroomCollector.addNewChatroom(newChatroom)

if WIAApp.currentChatroom == None:
WIAApp.currentChatroom =
WIAApp.chatroomCollector.getRoomByMemberID([WIAApp.clientInfo.ge
tID(), targe # Got the message while talking to another chatroom
twice = False
for room in WIAApp.chatroomCollector.getChatroomList(): if
(set(room.getMemberIDList()) == set(msg.getReceiverAddr()) and not
(WIAApp.clientInfo.getID() == msg.getOwnerID())):
room.addMessage(msg)

# If got any message while talking in the selected chatroom
if set(room.getMemberIDList()) ==
set(WIAApp.currentChatroom.getMemberIDList()) and not twice:
if msg.getOwnerID() != WIAApp.clientInfo.getID():
messageBox = MessageBoxPartner(msg.getOwnerName(),
msg.getText(), msg.getTimeCreated())
elif msg.getOwnerID() == WIAApp.clientInfo.getID():
messageBox = MessageBoxOwner("YOU", msg.getText(),
msg.getTimeCreated())
self.chatContainer.add_widget(messageBox) twice = True
WIAApp.mainUIScreen.updateHistoryType_2(msg)

def openChooserDialog(self):
cd = FileChooserDialog()
popup = Popup(title="Choose a file to send", content=cd, size_hint=
(.8, .8), auto_dismiss=True) popup.open()

```

```

def selectFile(self, path, filename):
    file = open(os.path.join(path, filename[0]), 'rb')

    fname = os.path.join(path, filename[0])
    fsize = os.path.getsize(fname)
    fname = fname.split("\\")[-1] # Use the last index as a filename

    obj = FileObject(fname, fsize, WIAApp.clientInfo.getID(),
[WIAApp.clientTargetID]) task = Task("Send File", obj)
    WIAApp.clientSocket.sendTask(task)

    data = file.read(1024)
    while data:
        print(">>Sending a file<<")
        WIAApp.clientSocket.sendData(data)
        data = file.read(1024)

    print("Completed sending file!") file.close()
class FileChooserDialog(BoxLayout): pass
class ProfileArea(BoxLayout): pass
class MenuBar(BoxLayout): pass
class ContactScreen(Screen):
    def __init__(self, **kwargs):
        super(Screen, self).__init__(**kwargs) class HistoryScreen(Screen):
    def __init__(self, **kwargs):
        super(Screen, self).__init__(**kwargs) def updateOrderChatList(self):
        pass
class RequestScreen(Screen):
    def __init__(self, **kwargs):
        super(Screen, self).__init__(**kwargs)

def responseInvitation(self, gname, creatorID, creatorName, answer,
root): inv = Invitation([creatorID], WIAApp.clientInfo, gname, answer)
task = Task("Response Invitation", inv)
WIAApp.clientSocket.sendTask(task)

if answer == "accept":
    # Create a groupchat widget

```

```
groupWidget = GroupChatComponent(gname, creatorID,  
creatorName) groupContainer =  
WlApp.mainUIScreen.screenSlider.contactScreen.groupContainer  
groupContainer.add_widget(groupWidget)
```

```
groupChatroom = Chatroom(gname, creatorID, rType="group")  
groupChatroom.addMemberID(creatorID)  
groupChatroom.addMemberID(WlApp.clientInfo.getID())  
WlApp.chatroomCollector.addNewChatroom(groupChatroom)
```

```
self.requestContainer.remove_widget(root) # Remove the request  
component
```

```
class ScreenSlider(Carousel): pass  
class ContactComponent(BoxLayout):
```

```
def __init__(self, id, name, **kwargs):  
BoxLayout.__init__(self) # This one must use syntax like this  
self.idButton.text = str(id)  
self.nameButton.text = name
```

```
class HistoryComponent(BoxLayout):  
def __init__(self, id, name, latestMsg, **kwargs):  
BoxLayout.__init__(self) # This one must use syntax like this  
self.idButton.text = str(id)  
self.nameButton.text = name  
self.latestMsg.text = latestMsg
```

```
class MessageBoxOwner(BoxLayout):  
def __init__(self, name, text, time, **kwargs): BoxLayout.__init__(self)  
self.senderArea.text = name  
self.textArea.text = text  
self.timeArea.text = time
```

```
class MessageBoxPartner(BoxLayout):  
def __init__(self, name, text, time, **kwargs): BoxLayout.__init__(self)  
self.senderArea.text = name  
self.textArea.text = text  
self.timeArea.text = time
```

```
#####  
##
```

```
class WIChat(ScreenManager):  
    def __init__(self, **kwargs):  
        super(WIChat, self).__init__(**kwargs) self.username = ""  
        self.ip = ""  
        self.port = ""  
        self.clientInfoList = None  
        self.historyInfoList = None  
        self.chatroomCollector = ChatroomCollector() self.serverSocket =  
        None  
        self.clientSocket = None  
        self.clientInfo = None  
        self.currentChatroom = None  
        self.clientTargetName = None  
        self.clientTargetID = None
```

```
lass WIChatApp(App): def build(self):  
    global WIApp  
    WIApp = WIChat()
```

```
    return WIApp  
    # App will not be terminated when open other Apps def  
    on_pause(self):  
        return True  
    WIChatApp().run()
```

chatroom.py

```
class Chatroom:  
    def __init__(self, roomName, creatorID = None, rType = "private",  
        **kwargs): self.roomName = roomName  
        self.creatorID = creatorID  
        self.rType = rType  
        self.msgCollector = [] ## Message obj  
        self.fileCollector = [] ## File obj  
        self.memberNameList = [] ## ClientName
```

```

self.memberIDList = [] ## ClientName
self.lengthOfData = 0

def getLengthOfData(self): return self.lengthOfData
def getWidgetCollector(self): return self.widgetCollector
def getRoomName(self): return self.roomName
def getCreatorID(self): return self.creatorID def getRoomType(self):
return self.rType def getMsgCollector(self): return self.msgCollector
def getFileCollector(self): return self.fileCollector
def getMemberNameList(self): return self.memberNameList
def getMemberIDList(self): return self.memberIDList
def setRoomName(self, name): self.roomName = name
def setRoomID(self, roomID): self.roomID = roomID
def addWidget(self, widget):
self.widgetCollector.append(widget)

def addMessage(self, msgObject):
self.msgCollector.append(msgObject) self.lengthOfData += 1

def addFile(self, fileObject):
self.fileCollector.append(fileObject) self.lengthOfData += 1

def addMemberID(self, newMemberID):
self.memberIDList.append(newMemberID)
def removeMsg(self, targetMsg):
for msg in self.msgCollector():
if msg.getText() == targetMsg.getText():
self.msgCollector.remove(msg)

def __str__(self):
s = "Room name: " + self.roomName + "\n" s += "Member: \n"
for id in self.memberIDList:

s += "\t" + id + "\n"
return s
def showAllMessageHistory(self): pass
ChatroomCollector.py
import random

```

```

class ChatroomCollector:
    def __init__(self, **kwargs):
        self.chatroomList = []
        self.roomIDs = []

    def getRoomByRoomName(self, roomName, creatorID): # For single chat
        for room in self.chatroomList:
            if room.getRoomType() == "group":
                if ( room.getRoomName() == roomName and room.getCreatorID() == creatorID ):
                    return room
            def getRoomByMemberName(self, memberNameList):
                for room in self.chatroomList:
                    if set(room.getMemberNameList()) == set(memberNameList):
                        return room
            def getRoomByMemberID(self, memberIDList):
                for room in self.chatroomList:
                    if room.getRoomType() == "single":
                        if set(room.getMemberIDList()) == set(memberIDList):
                            return room
            def getChatroomList(self):
                return self.chatroomList

    def addNewChatroom(self, newRoom):
        roomID = random.randint(150,400)
        while roomID in self.roomIDs:
            roomID = random.randint(150, 400)
        newRoom.setRoomID(roomID)
        self.chatroomList.append(newRoom)

    def listAllChatroom(self):
        for room in self.chatroomList:
            print(room)

    def __len__(self):
        return len(self.chatroomList)

    def isExist(self, gname, creatorID):
        for room in self.chatroomList:
            if ( room.getRoomName() == gname and room.getCreatorID() == creatorID ):
                return True
        return False

```

ClientCollector.py

```

class ClientCollector:
def __init__(self, **kwargs): self.clientInfoList = [] self.addressList = []
self.groupChatList = [] self.handlerList = []
self.socketList = []

def addSocket(self, clientSocket):
self.socketList.append(clientSocket)

def addClientInfo(self, clientInfo):
self.clientInfoList.append(clientInfo)
self.addressList.append(clientInfo.getAddress())

def showAllClients(self):
for eachClientInfo in self.clientInfoList: print(eachClientInfo)
def addGroupChat(self, group): self.groupChatList.append(group)
def addHandler(self, clientHandler):
self.handlerList.append(clientHandler)
def removeClientInfoByAddr(self, addr): for clientInfo in
self.clientInfoList: if clientInfo.getAddress() == addr:
self.clientInfoList.remove(clientInfo) def setNewStatus(self, id, status):
for clientInfo in self.clientInfoList: if clientInfo.getID() == id:
clientInfo.setStatus(status)
def removeSocket(self, soc): self.socketList.remove(soc)
def getClientInfoList(self): return self.clientInfoList

def getClientNameList(self): lst = []
for client in self.clientInfoList:

lst.append(client.getName())
return lst
def getSocketList(self): return self.socketList
def getAddrList(self): return self.addressList
def getAllClientInfo(self): s = "\n"
for client in self.clientInfoList: s += "\t" + str(client) + "\n"
return s
ClientInformation.py

```

```

class ClientInformation:
def __init__(self, name, status, address, profilePicture, task="",
**kwargs): self.id = address[1]
self.name = name
self.status = status
self.address = address
self.profilePicture = profilePicture
self.task = task
self.message = ""

def setTask(self, newTask): self.task = newTask
def setMessage(self, newMsg): self.message = newMsg
def setName(self, name): self.name = name
def setStatus(self, newStatus): self.status = newStatus
def setProfilePic(self, newPic): self.profilePicture = newPic
def getID(self):
return str(self.id)
def getTask(self): return self.task
def getName(self): return self.name def getStatus(self): return
self.status
def getProfilePic(self): return self.profilePicture
def getMessage(self): return self.message def getAddress(self):
return self.address
def __str__(self):
return self.name + " , id: " + str(self.address)

```

ClientSocket.py

```

import threading import socket import time
import pickle

```

```

from Task import *
from ClientInformation import * from Message import *

```

```

class ClientSocket:
def __init__(self, ip, port, name="Name", **kwargs): self.tLock =
threading.Lock()
self.shutdown = False
self.pauseMsg = True

```



```

self.clientName = name
self.clientMessage = ""
self.ip = ip
self.port = port
self.targetServer = (ip, port)

self.dataIncome = None

def connect(self):
self.soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.soc.connect(self.targetServer)
self.soc.setblocking(1)

self.recvThread = threading.Thread(target=self.receiving) ##
Receiving Thread self.recvThread.start()
def getAddr(self):
return self.soc.getsockname() def getDataIncome(self): return
self.dataIncome
def receiving(self):
while not self.shutdown:
try:
self.tLock.acquire()
self.dataIncome = self.soc.recv(1024)
except Exception as e: print(e)
finally:
self.tLock.release()
def setText(self, message): self.clientMessage = message
def startMessaging(self): self.pauseMsg = False
def pauseMessaging(self): self.pauseMsg = True

def sendTask(self, task): obj = pickle.dumps(task) self.soc.send(obj)

def sendData(self, data): self.soc.send(data)
def clearData(self):
self.dataIncome = None

def close(self):
self.shutdown = True self.recvThread.join() self.soc.close()

```

Contact.py

```
class Contact:
def __init__(self, name, status, id, **kwargs): self.name = name
self.status = status
self.id = id

def getName(self): return self.name
def getStatus(self): return self.status
def getID(self): return self.id
def __str__(self):
return "id: " + str(self.id) + ", " + self.name + ", state: " + self.status
```

FileObject.py

```
class FileObject:
def __init__(self, filename, fsize, ownerID, receiverAddrList,
**kwargs): self.filename = filename
self.fsize = fsize
self.ownerID = ownerID
self.receiverAddrList = receiverAddrList

def getFilename(self): return self.filename
def getFileSize(self): return self.fsize
def getOwnerID(self): return self.ownerID
def getReceiverAddr(self): return self.receiverAddrList
```

Handler.py

```
import pickle
import threading import time
import os

from ServerSocket import * from ClientCollector import * from Task
import *
from FileObject import *

class Handler(threading.Thread):
clientCollector = ClientCollector()
```

```
def __init__(self, soc, addr, **kwargs): threading.Thread.__init__(self)
self.soc = soc
self.addr = addr
self.exit = False
self.tLock = threading.Lock()
```

```
def notifiyAll(self):
for soc in self.clientCollector.getSocketList():
task_update_client = Task("Remove Client", self.soc.getpeername()
[1]) obj = pickle.dumps(task_update_client)
soc.send(obj)
```

```
def sendFileDetail(self, soc, directory, fileObj):
task = Task("FileDetail", fileObj)
obj = pickle.dumps(task)
```

```
def run(self):
try:
while not self.exit:
try:
data = self.soc.recv(1024)
if data:
task = pickle.loads(data)
if task.getName() == "Submit ClientInfo": clientInfo = task.getData()
if clientInfo.getAddress() not in self.clientCollector.getAddrList():
self.clientCollector.addClientInfo(clientInfo)
```

```
#### Add a new contact
for soc in self.clientCollector.getSocketList():
task_update_client = Task("New Client",
self.clientCollector.getClientInfoList())
obj = pickle.dumps(task_update_client)
soc.send(obj)
```

```
if task.getName() == "Message":
msgObject = task.getData()
receiverAddrs = msgObject.getReceiverAddr()
```

```

for soc in self.clientCollector.getSocketList():
for addr in receiverAddrs:
if self.soc != soc and soc.getpeername()[1] == int(addr): obj =
pickle.dumps(task)
soc.send(obj)

if task.getName() == "Group Message": msgObject = task.getData()
receiverAddrs = msgObject.getReceiverAddr()

for soc in self.clientCollector.getSocketList():
for addr in receiverAddrs:
if self.soc != soc and soc.getpeername()[1] == int(addr):

obj = pickle.dumps(task)
if task.getName() == "Update Group Members": groupChat =
task.getData()
receiverAddrs = groupChat.getMemberIDList()

for soc in self.clientCollector.getSocketList():
for addr in receiverAddrs:
if self.soc != soc and soc.getpeername()[1] == int(addr): obj =
pickle.dumps(task)
soc.send(obj)

if task.getName() == "Invite to group":
inviteObj = task.getData()
receiverAddrs = inviteObj.getReceiverAddr()
for soc in self.clientCollector.getSocketList():

for addr in receiverAddrs:
if self.soc != soc and soc.getpeername()[1] == int(addr): obj =
pickle.dumps(task)
soc.send(obj)

if task.getName() == "Response Invitation":
inviteObj = task.getData()
receiverAddrs = inviteObj.getReceiverAddr()
for soc in self.clientCollector.getSocketList():

```

```

for addr in receiverAddrs:
    if self.soc != soc and soc.getpeername()[1] == int(addr): obj =
    pickle.dumps(task)
    soc.send(obj)

if task.getName() == "Change Status":
    clientInfo = task.getData()
    for soc in self.clientCollector.getSocketList():

if self.soc != soc:
    task_update_client = Task("Change Status", clientInfo) obj =
    pickle.dumps(task_update_client)
    soc.send(obj)

if task.getName() == "Send File":

    obj = task.getData()
    receiverAddrs = obj.getReceiverAddr()
    for soc in self.clientCollector.getSocketList():

for addr in receiverAddrs:
    if self.soc != soc and soc.getpeername()[1] == int(addr): filename =
    obj.getFilename()
    filesize = obj.getFileSize()
    directory = "serverFiles/" + filename
    file = open(directory, "wb")
    data = self.soc.recv(1024)
    targetSize = filesize
    currentSize = 0
    if filesize <= 1024:

file.write(data)
elif filesize > 1024:
    while filesize >= 0:
    print(">>Receiving a file : ", str(100*currentSize//targetSize) + " %
    <<") file.write(data)
    currentSize += 1024
    filesize -= 1024

```

```

if filesize < 0:
    break
data = self.soc.recv(1024)

file.close()
print("Server received a new file.") print("Closed file.")

self.sendFileDetail(soc, directory, obj)

except OverflowError:
    pass
except ValueError:
    pass
except KeyError:
    pass
except EOFError:
    pass
except pickle.UnpicklingError:
    pass
except pickle.PicklingError:
    pass
except pickle.PickleError:
    pass

except ConnectionResetError:
    print(self.addr, " has been disconnected")
    self.exit = True
    self.clientCollector.removeSocket(self.soc)
    self.clientCollector.removeClientInfoByAddr(self.soc.getpeername())
    self.notifyAll()
    self.soc.close()

```

Invitation.py

```

class Invitation:
    def __init__(self, receivedAddrs, ownerInfo, groupName, response =
False, **kwargs): self.receivedAddrs = receivedAddrs
    self.ownerInfo = ownerInfo

```

```
self.groupName = groupName
self.response = response
```

```
def getReceiverAddr(self): return self.receiverAddr
def getOwnerInfo(self): return self.ownerInfo
def getGroupName(self): return self.groupName
def getResponse(self): return self.response
```

Message.py

```
import time
from datetime import datetime
```

```
class Message:
def __init__(self, text, receiverAddrList, ownerInfo, groupName =
None, roomCreatorID = None,
timeCreated=None, **kwargs):
self.text = text
self.receiverAddr = receiverAddrList # Socket address of the receiver
self.ownerInfo = ownerInfo
self.groupName = groupName
self.roomCreatorID = roomCreatorID
self.ownerName = ownerInfo[0]
self.ownerID = ownerInfo[1]
self.timeCreated = timeCreated
```

```
def getOwnerInfo(self): return self.ownerInfo def
def getOwnerName(self): return self.ownerName
def getOwnerID(self): return self.ownerID
def getText(self): return self.text
def getReceiverAddr(self): return self.receiverAddr
def getGroupName(self): return self.groupName
def getRoomCreatorID(self): return self.roomCreatorID
def getTimeCreated(self): return self.timeCreated
```

ServerSocket.py

```
import threading import socket import time
```

```
from Handler import *
from ClientCollector import ClientCollector
```

```

class ServerSocket(threading.Thread):
    def __init__(self, ip, port, maximumClient = 10, **kwargs):
        threading.Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.server = ((ip, port))
        self.maximumClient = maximumClient
        self.clientCollector = ClientCollector()
        self.setDaemon(True)

    def run(self):
        self.startServer()
        def startServer(self):

            try:
                self.socketServer = socket.socket(socket.AF_INET,
                socket.SOCK_STREAM) self.socketServer.bind(self.server)
                self.socketServer.listen(self.maximumClient)
                self.socketServer.setblocking(1)
                print("Server is ready for connection.")
                self.listen()

            except OSError as e:
                print("\n<<< Server is already working >>>")
                def listen(self):
                    try:
                        while True:
                            clientSocket, addr = self.socketServer.accept()

                            handler = Handler(clientSocket, addr)
                            handler.setDaemon(True)
                            handler.start()
                            handler.clientCollector.addSocket(clientSocket)
                            handler.clientCollector.addHandler(handler)

                            time.sleep(0.1) ## Wait a bit for updating value in a Handler object
                    except OSError:
                        print("The port is not available")

```

Task.py


```
class Task:
def __init__(self, taskName, data, **kwargs): self.taskName =
taskName
self.data = data
```

```
def getName(self): return self.taskName
def getData(self): return self.data
```

WChat.kv

```
#:import SwapTransition kivy.uix.screenmanager.SwapTransition
#:import FadeTransition kivy.uix.screenmanager.FadeTransition
#:import SlideTransition kivy.uix.screenmanager.SlideTransition
```

```
<WChat>:
transition: SlideTransition(direction="up") startupScreen:
StartupScreen
mainUIScreen: MainUIScreen
chatroomScreen: ChatroomScreen createGroupScreen:
CreateGroupScreen groupchatScreen: GroupChatScreen
```

```
StartupScreen:
id: StartupScreen
MainUIScreen:
id: MainUIScreen
ChatroomScreen:
id: ChatroomScreen
CreateGroupScreen: id: CreateGroupScreen
GroupChatScreen:
id: GroupChatScreen
```

```
<StartupScreen>:
name: "StartupScreen"
```

```
nameInput: nameInput ipInput: ipInput
portInput: portInput
```

```
BoxLayout:
orientation: 'vertical' spacing: 15
```

canvas.before:

Color:

rgba: 0.157, 0.643, 1.000, 1

Rectangle:

pos: self.pos

size: self.size

Label:

id: StartupLabel text: "Start Up" size_hint: 1, .1 font_size: '30sp'

canvas.before:

Color:

rgba: 0.027, 0.412, 0.698, 1

Rectangle:

pos: self.pos

size: self.size

BoxLayout:

orientation: 'vertical' size_hint_y: 0.2

Label:

text: "NICKNAME" halign: 'center' size_hint_y: .1 font_size: '25sp'

TextInput:

id: nameInput

text: "Untitled"

font_size: '20sp'

padding_x: self.width/4

multiline: False

input_filter: lambda text, from_undo: text[:8 - len(self.text)] size_hint:
.5, .1

pos_hint: {'x': .25}

BoxLayout:

orientation: 'vertical' size_hint_y: 0.2

Label:

text: "HOST IP"

size_hint_y: .1
font_size: '25sp'

TextInput:
id: ipInput
text: "127.0.0.1"
font_size: '20sp'
multiline: False
size_hint: .5, .1
pos_hint: {'x': .25}
padding_x: self.width/4

BoxLayout:
orientation: 'vertical' size_hint_y: 0.2

Label:
text: "PORT" size_hint_y: .1 font_size: '25sp'

TextInput:
id: portInput
text: "5000"
font_size: '20sp'
multiline: False
size_hint: .5, .1
pos_hint: {'x': .25}
padding_x: self.width/4

Label:
size_hint_y: 0.2

BoxLayout:
orientation: 'horizontal' size_hint: 1, .1

Button:
id: loginButton
text: "LOG IN AS CLIENT" font_size: "16sp"
on_press: root.login()

background_normal: " id: loginButton
text: "BECOME A HOST"
font_size: "16sp"
on_press: root.openHostPopup()

background_normal: "
background_color: 0.000, 0.314, 0.541, 1
<ProfileArea>:
orientation: 'horizontal'

idButton: idButton
nameButton: nameButton statusButton: statusButton size_hint: (1, .2)
padding: 8

canvas.before:

Color:
rgba: 0.027, 0.412, 0.698, 1
Rectangle:
pos: self.pos
size: self.size

Button:
id: nameButton
size_hint_x: .3
font_size: '15sp'
background_normal: 'pictures/backgroundPic.png' background_color:
0.196, 0.663, 0.914, 1 border: 30,30,30,30

BoxLayout:
padding: 8
orientation: 'vertical'

Button:
id: idButton
text: "ID"
pos_hint: {'x': .125} size_hint_x: .8
size_hint_y: .2

```
background_normal: " id: statusButton
text: "Status"
pos_hint: {'x': .125}
size_hint_x: .8
size_hint_y: .2
background_normal: "
background_color: 0.196, 0.663, 0.914, 1
```

```
on_press: app.root.mainUIScreen.openStatusInput()
```

```
<MenuBar>:
orientation: 'horizontal'
contactButton: contactButton historyButton: historyButton
requestButton: requestButton settingButton: settingButton size_hint:
(1, .1)
```

```
Button:
text: 'Contact'
id: contactButton
background_normal: 'pictures/menubarBG_normal.png'
background_down: 'pictures/menubarBG_selected.png' border:
30,30,30,30
```

```
on_press: app.root.mainUIScreen.changeScreen('contact')
```

```
Button:
text: 'History'
id: historyButton
background_normal: 'pictures/menubarBG_normal.png'
background_down: 'pictures/menubarBG_selected.png'
```

```
on_press: app.root.mainUIScreen.changeScreen('history')
```

```
Button:
text: 'Request'
id: requestButton
background_normal: 'pictures/menubarBG_normal.png'
background_down: 'pictures/menubarBG_selected.png'
```

on_press: app.root.mainUIScreen.changeScreen('request') text:
'Setting'

id: settingButton

background_normal: 'pictures/menubarBG_normal.png'

background_down: 'pictures/menubarBG_selected.png'

<GroupChatComponent>:

orientation: 'horizontal'

gnameButton: gnameButton creatorID: creatorIDLabel

creatorName: creatorNameLabel size_hint: None, None

width: app.root.mainUIScreen.width

primary_color: 0.000, 0.314, 0.541, 1 secondary_color: 0.020, 0.584,
0.647, 1

Button:

id: gnameButton

size_hint: .3, 1

font_size: '15sp'

background_normal: ''

background_color: root.primary_color

on_press:

app.root.mainUIScreen.moveto_groupchat(gnameButton.text,
root.creatorID.text)

BoxLayout:

size_hint: .7, 1

orientation: 'vertical'

Label:

text: 'Group Chat'

font_size: '16sp'

canvas.before:

Color:

rgba: root.secondary_color

Rectangle:

pos: self.pos

size: self.size
id: creatorIDLabel
font_size: '16sp'
canvas.before:
Color:
rgba: root.secondary_color
Rectangle:
pos: self.pos
size: self.size

Label:
id: creatorNameLabel
font_size: '16sp'
canvas.before:

Color:
rgba: root.secondary_color
Rectangle:
pos: self.pos
size: self.size

<ContactComponent>:
orientation: 'horizontal'
size_hint: None, None
width: app.root.mainUIScreen.width

idButton: idButton
nameButton: nameButton
statusButton: statusButton

Button:
background_normal: "
background_color: 0.196, 0.663, 0.914, 1 size_hint: .3, 1
font_size: '15sp'
id: nameButton

on_press: app.root.mainUIScreen.moveto_chatroom(idButton.text,
nameButton.text)

BoxLayout:
orientation: 'vertical' id: idButton
font_size: '16sp'
canvas.before:

Color:
rgba: 0.369, 0.545, 0.776, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: statusButton text: "None"
font_size: '16sp'

canvas.before:

Color:
rgba: 0.369, 0.545, 0.776, 1
Rectangle:
pos: self.pos
size: self.size

<ContactScreen>:
orientation: 'vertical'
contactScrollView: contactScrollView groupContainer:
groupContainer

Accordion:
orientation: 'vertical'

AccordionItem:
title: 'Available Groups'
background_selected: 'pictures/accordionBG_normal.png'
background_normal: 'pictures/accordionBG_selected.png' id:
groupContainer
cols: 1
padding: 2.5

spacing: 5
size_hint: 1, None
height: self.minimum_height

Button:

text: "++ Create a new group ++"
font_size: "18sp"
size_hint: 1, None
on_press: app.root.mainUIScreen.moveto_createGroup()

background_normal: "
background_color: 0.000, 0.475, 0.824, 1

AccordionItem:

title: 'Available Contacts'
background_selected: 'pictures/accordionBG_normal.png'
background_normal: 'pictures/accordionBG_selected.png'

ScrollView:

size: self.parent.width, self.parent.height size_hint: None, None
GridLayout:

id: contactScrollView
cols: 1
padding: 2.5
spacing: 5
size_hint: 1, None
height: self.minimum_height

<HistoryScreen>:

orientation: 'vertical'
historyScrollView: historyScrollView

ScrollView:

size: self.parent.width, self.parent.height size_hint: None, None
GridLayout:

id: historyScrollView
cols: 1

padding: 2.5
spacing: 5
size_hint: None, None
height: self.minimum_height
<RequestScreen>:

orientation: 'vertical'
requestContainer: requestContainer

ScrollView:
size: self.parent.width, self.parent.height size_hint: None, None
GridLayout:

id: requestContainer
cols: 1
padding: 2.5
spacing: 5
size_hint: 1, None
height: self.minimum_height

<HistoryGroupComponent>:
orientation: 'horizontal'
gnameButton: gnameButton
creatorID: creatorIDLabel
lastestMsg: lastestMsg
size_hint: None, None
width: app.root.mainUIScreen.width

primary_color: 0.000, 0.314, 0.541, 1 secondary_color: 0.020, 0.584,
0.647, 1

Button:
id: gnameButton
size_hint: .3, 1
background_normal: "
background_color: root.primary_color
on_press:

```
app.root.mainUIScreen.moveto_groupchat(gnameButton.text,  
root.creatorID.text)
```

BoxLayout:

orientation: 'vertical'

Label:

id: creatorIDLabel

canvas.before:

Color:

rgba: root.secondary_color

Rectangle:

pos: self.pos

id: lastestMsg

canvas.before:

Color:

rgba: root.secondary_color Rectangle:

pos: self.pos

size: self.size

<HistoryComponent>:

orientation: 'horizontal'

size_hint: None, None

width: app.root.mainUIScreen.width

idButton: idButton

nameButton: nameButton lastestMsg: lastestMsg

Button:

id: nameButton

background_normal: "

background_color: 0.196, 0.663, 0.914, 1

size_hint: .3, 1

on_press: app.root.mainUIScreen.moveto_chatroom(idButton.text,
nameButton.text)

BoxLayout:
orientation: 'vertical'
Label:

id: idButton
canvas.before:

Color:
rgba: 0.369, 0.545, 0.776, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: lastestMsg
text: "None"
canvas.before:

Color:
rgba: 0.369, 0.545, 0.776, 1
Rectangle:
pos: self.pos
<ScreenSlider>:
anim_move_duration: .3 anim_cancel_duration: .5

canvas.before:

Color:
rgba: 0.804, 0.804, 0.804, 1
Rectangle:
pos: self.pos
size: self.size

contactScreen: contactScreen historyScreen: historyScreen
requestScreen: requestScreen

ContactScreen:
id: contactScreen
HistoryScreen:

id: historyScreen
RequestScreen:
id: requestScreen

<InAppNotification>: orientation: 'horizontal' title: title
detail: detail
size_hint_y: .1

Label:
id: title
color: 1, 1, 1, 1 size_hint: .3,1

canvas.before:

Color:
rgba: 0.000, 0.643, 0.580, 1
Rectangle:
pos: self.pos
id: detail
color: 0,0,0,1 size_hint: .7,1

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

<RequestComponent>:
orientation: 'horizontal' nameButton: nameButton idButton: idButton
groupButton: groupButton acceptButton: acceptButton cancelButton:
cancelButton size_hint: 1, None

BoxLayout:
orientation: 'vertical' size_hint: .6, 1

Label:
id: groupButton color: 1, 1, 1 ,1 size_hint: 1,.4 font_size: '16sp'

canvas.before:

Color:

rgba: 0.000, 0.643, 0.580, 1

Rectangle:

pos: self.pos

size: self.size

Label:

id: idButton

color: 0,0,0,1

size_hint: 1,.3

canvas.before:

Color:

rgba: 0.894, 0.910, 0.922, 1

Rectangle:

pos: self.pos

size: self.size

Label:

id: nameButton color: 0,0,0,1 size_hint: 1,.3 font_size: '16sp'

canvas.before:

Color:

rgba: 0.894, 0.910, 0.922, 1

Rectangle:

pos: self.pos

size: self.size

BoxLayout:

orientation: 'vertical' size_hint: .08, 1

Button:

id: acceptButton size_hint: 1, .5 text: 'Y'

font_size: '16sp'

on_press:
app.root.mainUIScreen.screenSlider.requestScreen.responseInvitation(groupButton.text, idButton.text, nameButton.text, "accept", root)

background_normal: "
background_color: 0.000, 0.643, 0.580, 1

Button:
id: cancelButton size_hint: 1, .5 text: 'N'
on_press:

app.root.mainUIScreen.screenSlider.requestScreen.responseInvitation(groupButton.text, idButton.text, nameButton.text, "cancel", root)
background_normal: "
background_color: 0.000, 0.643, 0.580, 1

<MainUIScreen>:
name: "MainUIScreen"
profileArea: profileArea
menuBar: menuBar
screenSlider: screenSlider
inAppNotification: inAppNotification

FloatLayout:

BoxLayout:
orientation: 'vertical' ProfileArea:

id: profileArea

MenuBar:
id: menuBar
ScreenSlider:
id: screenSlider

BoxLayout:
id: inAppNotification
pos: -self.parent.width, self.parent.height - self.height size_hint: 1, .1
orientation: 'horizontal'

title: title
detail: detail

Label:
id: title
color: 1,1,1,1 size_hint: .3,1

canvas.before:

Color:
rgba: 0.000, 0.643, 0.580, 1
Rectangle:
pos: self.pos
Label:
id: detail
color: 0,0,0,1 size_hint: .7,1

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

<MessageBoxOwner>: textArea: textArea
timeArea: timeArea senderArea: senderArea size_hint: 1, None
orientation: 'horizontal'

BoxLayout:
orientation: 'vertical' size_hint: 1, None

Label:
id: textArea
text: 'text message' size_hint_y: 0.8 color: 0,0,0,1
font_size: '16sp'

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: timeArea
text: 'time'
size_hint_y: 0.2
color: 0,0,0,1

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: senderArea
text: 'YOU'
size_hint: 0.3, None font_size: '16sp'

canvas.before:

Color:
rgba: 0.192, 0.263, 0.592, 1
Rectangle:
pos: self.pos
size: self.size

<MessageBoxPartner>: textArea: textArea
timeArea: timeArea senderArea: senderArea size_hint: 1, None

BoxLayout:
orientation: 'horizontal' height: 400

Label:
id: senderArea
text: 'Partner'
color: 1, 1, 1, 1
size_hint: 0.3, None
font_size: '16sp'

canvas.before:

Color:
rgba: 0.000, 0.643, 0.580, 1
Rectangle:
pos: self.pos
BoxLayout:
orientation: 'vertical' size_hint: 1, None

Label:
id: textArea
text: 'text message' size_hint_y: 0.8 color: 0,0,0,1
font_size: '16sp'

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: timeArea
text: 'time'
size_hint_y: 0.2 color: 0,0,0,1 font_size: '13sp'

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:

pos: self.pos
size: self.size

<FileChooserDialog>:
orientation: 'vertical'
chooser: filechooser

FileChooserListView: id: filechooser
path: "./"

BoxLayout:
orientation: 'horizontal' size_hint_y: 0.1

Button:
text: 'SELECT'
background_normal: "
background_color: 0.071, 0.4, 0.840, 1

on_press: app.root.chatroomScreen.selectFile(root.chooser.path,
root.chooser.selection)

<InviteComponent>:
orientation: 'horizontal'
size_hint: None, None
selection: selection
width: app.root.mainUIScreen.width idButton: idButton
nameButton: nameButton
CheckBox:

id: selection
size_hint: .1, None
color: 0,0,.400,1

canvas.before:

Color:
rgba: 0.102, 0.314, 0.549, 1
Rectangle:
pos: self.pos

size: self.size
BoxLayout:
orientation: 'vertical'
Label:
id: idButton
text: "ID"
font_size: '16sp'
color: 0,0,0,1
size_hint: 1, .5
canvas.before:
Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size
Label:
id: nameButton
text: "Name"
font_size: '16sp'
color: 0,0,0,1
<CreateGroupScreen>:
name: "CreateGroupScreen" contactContainer: contactContainer
groupNameInput: groupNameInput

BoxLayout:
orientation: 'vertical'

BoxLayout:
orientation: 'horizontal' size_hint: 1, .1

Label:
text: 'Invite contacts to group' size_hint_x: .8

canvas.before:

Color:
rgba: 0.369, 0.545, 0.776, 1
Rectangle:

pos: self.pos
size: self.size

Button:
background_normal: "
background_color: 0.008, 0.267, 0.463, 1
text: "Back"
size_hint_x: .2
on_press: app.root.chatroomScreen.moveto_mainUI()

BoxLayout:
orientation: 'horizontal' size_hint: 1, .1

Label:
text: 'Group Name:'
canvas.before:

Color:
rgba: 0, 0, .600, 1
Rectangle:
pos: self.pos
size: self.size

TextInput:
id: groupNameInput
text: '>> Input <<'
size_hint: .7, 1
multiline: False
input_filter: lambda text, from_undo: text[:20 - len(self.text)]

ScrollView:
size: self.parent.width, self.parent.height do_scroll_x: False
bar_width: 10
bar_color: (0.000, 0.643, 0.616, 1)

canvas.before:

Color:
rgba: 0.804, 0.804, 0.804, 1

Rectangle:
pos: self.pos
size: self.size

GridLayout:
id: contactContainer
cols: 1
padding: 2.5
spacing: 5
size_hint: 1, None
height: self.minimum_height

BoxLayout:
orientation: 'horizontal'
size_hint: 1, .1

Button:
text: 'Select'
id: 'selectButton'
size_hint: .5, 1
on_press: root.sendInvitation() background_normal: "
background_color: 0.000, 0.314, 0.541, 1

Button:
text: 'CANCEL'
id: 'cancelButton'
size_hint: .5, 1
on_press: root.moveto_mainUI()

background_normal: "
background_color: 0.000, 0.314, 0.541, 1

<IDContainer>:
container: container orientation: 'vertical'

ScrollView:
size: self.parent.width, self.parent.height do_scroll_x: False

bar_width: 2
bar_color: (0.000, 0.643, 0.616, 1)

GridLayout:
id: container
cols: 1
padding: 10
spacing: 10
size_hint: 1, None
height: self.minimum_height

<GroupChatScreen>:
name: "GroupChatScreen"
groupchatContainer: groupchatContainer messageInput:
messageInput
roomName: roomName
inRoomNotification: inRoomNotification menu: menu

FloatLayout:
BoxLayout:
BoxLayout:
orientation: 'vertical'
BoxLayout:
orientation: "horizontal"

Button:
background_normal: "
background_color: 0.369, 0.545, 0.776, 1

id: roomName text: "Chatroom" size_hint_x: .6

Button:
background_normal: 'pictures/backButton.png' background_down:
'pictures/backButton_selected.png' size_hint_x: .2
on_press: app.root.chatroomScreen.moveto_mainUI()

Button:
background_normal: 'pictures/menulcon.png' background_down:

'pictures/menulcon_selected.png' size_hint_x: .2
on_press: root.showMenu()

ScrollView:

id: scrollView_chat

size: self.parent.width, self.parent.height do_scroll_x: False

bar_width: 10

bar_color: (0.000, 0.643, 0.616, 1)

canvas.before:

Color:

rgba: 0.804, 0.804, 0.804, 1

Rectangle:

pos: self.pos

size: self.size

GridLayout:

id: groupchatContainer

cols: 1

padding: 2.5

spacing: 5

size_hint: 1, None

height: self.minimum_height canvas.before:

Color:

rgba: 0.565, 0.663, 0.840, 1

Rectangle:

pos: self.pos

size: self.size

padding: 6

orientation: 'horizontal' size_hint: 1, .1

Button:

background_normal: "

background_color: 0.071, 0.4, 0.840, 1

id: cfButton
text: 'Send File'
size_hint: .25, 1
on_press: root.openChooserDialog()

TextInput:
id: messageInput
size_hint: .4, 1
multiline: False
input_filter: lambda text, from_undo: text[:26 - len(self.text)] on_focus:
root.hideMenu()

Button:
background_normal: "
background_color: 0.071, 0.4, 0.840, 1

id: sendButton
text: "Send"
size_hint: .2, 1
on_release: app.root.clientSocket.setText(messageInput.text)
on_release: root.sendMessageTask(root.roomName.text,
root.roomCreatorID)

BoxLayout:
id: inRoomNotification
pos: -self.parent.width, self.parent.height - self.height
size_hint: 1, .1
orientation: 'horizontal'
title: title
detail: detail

Label:
id: title
color: 1,1,1,1 size_hint: .3,1

canvas.before:

Color:
rgba: 0.000, 0.643, 0.580, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: detail
color: 0,0,0,1 size_hint: .7,1

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

BoxLayout:
id: menu
pos: -self.parent.width, 0 size_hint_x: 0.6

orientation: 'vertical' canvas.before:
Color:
rgba: 0.000, 0.314, 0.541, 1

Rectangle:
pos: self.pos size: self.size

Button:
size_hint: 1, 0.08
text: "Setting Panel"
background_normal: "
background_color: 0.027, 0.412, 0.698, 1 orientation: 'vertical'
size_hint: 1, 0.9
spacing: 5

Button:
text: "List Members" size_hint_y: .1

background_normal: "
background_color: (0.000, 0.643, 0.616, 1)
on_press: root.listMembers() on_press: root.hideMenu()

Button:
text: "Backup as text file" size_hint_y: .1

background_normal: "
background_color: (0.000, 0.643, 0.616, 1)
on_press: root.hideMenu() on_press: root.backup()

Button:
text: "Delete History Messages" size_hint_y: .1

background_normal: "
background_color: (0.000, 0.643, 0.616, 1)
on_press: root.hideMenu()
on_press: root.groupchatContainer.clear_widgets()

Label:
size_hint_y: .6

<ChatroomScreen>:
name: "ChatroomScreen"
chatContainer: chatContainer
messageInput: messageInput
roomName: roomName
menu: menu
inRoomNotification: inRoomNotification FloatLayout:

BoxLayout:
BoxLayout:
orientation: 'vertical'

BoxLayout:
orientation: "horizontal" size_hint: 1, .1

Button:
background_normal: "
background_color: 0.369, 0.545, 0.776, 1

id: roomName text: "Chatroom" size_hint_x: .6

Button:

background_normal: 'pictures/backButton.png' background_down:
'pictures/backButton_selected.png' size_hint_x: .2
on_press: app.root.chatroomScreen.moveto_mainUI()

Button:

background_normal: 'pictures/menulcon.png' background_down:
'pictures/menulcon_selected.png' size_hint_x: .2
on_press: root.showMenu()

ScrollView:

id: scrollView_chat
size: self.parent.width, self.parent.height
bar_width: 10
bar_color: (0.000, 0.643, 0.616, 1)

canvas.before:

Color:

rgba: 0.804, 0.804, 0.804, 1

Rectangle:

pos: self.pos

GridLayout:

id: chatContainer

cols: 1

padding: 2.5

spacing: 5

size_hint: 1, None

height: self.minimum_height

BoxLayout:

canvas.before:

Color:

rgba: 0.565, 0.663, 0.840, 1

Rectangle:

pos: self.pos
size: self.size

padding: 6
orientation: 'horizontal' size_hint: 1, .1

Button:
background_normal: "
background_color: 0.071, 0.4, 0.840, 1

id: cfButton
text: 'Send File'
size_hint: .25, 1
on_press: root.openChooserDialog()

TextInput:
id: messageInput
size_hint: .4, 1
multiline: False
input_filter: lambda text, from_undo: text[:26 - len(self.text)] on_focus:
root.hideMenu()

Button:
background_normal: "
background_color: 0.071, 0.4, 0.840, 1

id: sendButton
text: "Send"
size_hint: .2, 1
on_release: app.root.clientSocket.setText(messageInput.text)
on_release: root.sendMessageTask()

BoxLayout:
id: inRoomNotification
pos: -self.parent.width, self.parent.height - self.height size_hint: 1, .1
orientation: 'horizontal'
title: title
detail: detail

Label:
id: title
color: 1,1,1,1 size_hint: .3,1

canvas.before:

Color:
rgba: 0.000, 0.643, 0.580, 1
Rectangle:
pos: self.pos
size: self.size

Label:
id: detail
color: 0,0,0,1 size_hint: .7,1

canvas.before:

Color:
rgba: 0.894, 0.910, 0.922, 1
Rectangle:
pos: self.pos
size: self.size

BoxLayout:
id: menu
pos: -self.parent.width, 0 size_hint_x: 0.6

orientation: 'vertical'
canvas.before:
Color:
rgba: 0.000, 0.314, 0.541, 1
Rectangle:
pos: self.pos

Button:
size_hint: 1, 0.08
text: "Setting Panel"

```
background_normal: "  
background_color: 0.027, 0.412, 0.698, 1
```

```
BoxLayout:  
orientation: 'vertical' size_hint: 1, 0.9  
spacing: 5
```

```
Button:  
text: "Backup" size_hint_y: .1
```

```
background_normal: "  
background_color: 0.000, 0.643, 0.616, 1  
on_press: root.hideMenu() on_press: root.backup()
```

```
Button:  
text: "Delete History Messages" size_hint_y: .1
```

```
background_normal: "  
background_color: 0.000, 0.643, 0.616, 1  
on_press: root.hideMenu()  
on_press: root.chatContainer.clear_widgets()  
Label:  
size_hint_y: .6
```

This should return your chat app, it will let you connect to the server and start chatting. this was the longest app that we did in this book, so it will take you time to understand it all, but look at it in bits and pieces and try to understand the actual code first

if you have read and understood the codes given in chapter 7, then you are probably ready to start building your own apps. if you have not, keep revising it and going back to the other chapters for revision of what you don't understand and relearn the basics

Chapter eight : Whats next?

If you have read the book this far, you have actually achieved a lot for yourself. I hope you enjoyed reading the book. Now that you are an

app developer, I recommend you start on your own project ASAP. You will not be perfect on your first few projects, so always keep the book for reference in case you forget something.

How to develop a great app:

If you want to build a great app, build an app that solves a problem in the world, it will not be easy, but your hard work will eventually pay off. Always prioritize your users, because it is them that decide if your app is great or not.

BEST OF LUCK!