

# BCS-094 Programming using Python



# MCS-094

## PROGRAMMING USING PYTHON

IGNOU is one of the participants for the development of courses for this Programme.

### Copyright

This course has been developed as part of the collaborative advanced ICT course development project of the Commonwealth of Learning (COL). COL is an intergovernmental organisation created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources and technologies.

The Open University of Sri Lanka (OUSL) is the premier Open and Distance learning institution in the country where students can pursue their studies through Open and Distance Learning (ODL) methodologies. Degrees awarded by OUSL are treated as equivalent to the degrees awarded by other national universities in Sri Lanka by the University Grants Commission of Sri Lanka.



The Python Software Foundation is owner of Python. Copyright © 2001-2019 Python Software Foundation; All Rights Reserved. Reference: <https://docs.python.org/>

Kivy is an open source Python library for development of applications. Reference: <https://kivy.org/>

This Programming with Python is a teaching learning resource developed by the Open University of Sri Lanka using various sources duly acknowledged.

© Open University of Sri Lanka and Commonwealth of Learning, 2018. Except where otherwise noted, The Programming with Python course material is made available under Creative Commons AttributionShareAlike 4.0 International (CC BY-SA 4.0) License: <https://creativecommons.org/licenses/by-sa/4.0/legalcode>.

For the avoidance of doubt, by applying this license the Commonwealth of Learning does not waive any privileges or immunities from claims that it may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes or laws of any jurisdiction. The ideas and opinions expressed in this publication are those of the author/s; they are not necessarily those of Commonwealth of Learning and do not commit the organisation.



The Open University of Sri Lanka  
P. O. Box 10250, Nawala, Nugegoda  
Sri Lanka  
Phone: +94 112881481  
Fax: +94112821285  
Email: [hdelect@ou.ac.lk](mailto:hdelect@ou.ac.lk)  
Website: [www.ou.ac.lk](http://www.ou.ac.lk)



Commonwealth of Learning  
4710 Kingsway, Suite 2500, Burnaby  
V5H 4M2, British Columbia, Canada  
Phone: +1 604 775 8200  
Fax: +1 604 775 8210  
Email: [info@col.org](mailto:info@col.org)  
Website: [www.col.org](http://www.col.org)

---

## ACKNOWLEDGEMENT BY AUTHORS

---

The Open University of Sri Lanka (OUSL), Department of Electrical and Computer Engineering (ECE) wishes to thank those below for their contribution to this course material and accompanying videos and screencasts:

---

### AUTHORS

---

Unit 11 NT De Silva (Project Assistant, Dept. of ECE, OUSL)

Units 12 and 13 GSN Meedin (Lecturer, Dept. of ECE, OUSL)

Units 14 and 15 MHMND Herath (Lecturer, Dept. of ECE, OUSL)

---

#### Content Editor

DN Koggalahewa (Senior Lecturer, Sri Lanka Institute of IT)

#### Language Editor:

KARD Gunaratne (Lecturer, Dept. of ECE, OUSL)

#### Reviewer:

KARD Gunaratne (Lecturer, Dept. of ECE, OUSL)

---

#### Video presenters:

S Rajasingham (Lecturer, Dept. of ECE, OUSL) M Abeykoon (Software Engineer, Duo Software) SN Muhandiram (Associate Software Engineer, VizuaMatix )

#### Screencasters:

MHMND Herath (Lecturer, Dept. of ECE, OUSL) M Abeykoon (Software Engineer, Duo Software)

---

### COURSE COORDINATOR

---

Dr. Sudhansh Sharma,  
Assistant Professor  
School of Computers and Information Sciences(SOCIS)  
Indira Gandhi National Open University(IGNOU),  
New Delhi-110068

In this Block the Introduction to the Block along with the Check Your Progress in all the units, and Answers to Check Your Progress Questions are written and added in 2021, By

Dr. Sudhansh Sharma, Assistant Professor  
SOCIS, IGNOU, New Delhi-110068

---

### PRINT PRODUCTION

---

Mr. Tilak Raj  
Assistant Registrar (Publication)  
MPDD, IGNOU, New Delhi

Mr. Yashpal  
Assistant Registrar (Publication)  
MPDD, IGNOU, New Delhi

April, 2021

© Indira Gandhi National Open University, 2021

ISBN-

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information, about the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110 068.

Printed and published on behalf of the Indira Gandhi National Open University by Registrar, MPDD, IGNOU, New Delhi

Laser Composed by Tessa Media & Computers, C-206, Shaheen Bagh, Jamia Nagar, New Delhi

# BLOCK

# 3

## DATA HANDLING, MOBILE APPLICATION AND GUI DEVELOPMENT USING PYTHON

---

### UNIT 11

Handling Data with Python	5
---------------------------	---

---

### UNIT 12

Role of Python in Mobile Application Development	17
--	----

---

### UNIT 13

Mobile Application Development with Python	29
--	----

---

### UNIT 14

Python Graphical User Interface Development	35
---	----

---

### UNIT 15

GUI Programming Using KIVY Libraries	56
--------------------------------------	----

---

---

## BLOCK INTRODUCTION

---

Python is the most popular high-level programming language that can be used for real-world programming. It is a dynamic and object-oriented programming language that can be used in a vast domain of applications, especially data handling. Python was designed as a very user-friendly general-purpose language with a collection of modules and packages and gained popularity in the recent times. Whenever we think of machine learning, big data handling, data analysis, statistical algorithms, python is the only name which comes first in our mind. It is a dynamically typed language, which makes it highly flexible. Furthermore, creating a database and linking of the database is very fast and secure. It makes the program to compile and run in an extremely flexible environment can support different styles of programming, including structural and object-oriented. Its ability to use modular components that were designed in other programming languages allows a programmer to embed the code written in python for any interface. Due to its enormous features, python has become the first choice of learners in the field of data science and machine learning.

The content present in this block makes you to learn about the importance of databases in real world applications and importance of using SQLite with Python. Firstly, you will learn about integrating database concepts into real world applications and data management using SQLite. Next, you will learn how to use these different techniques and when to use them. On completion of the content, you will be able to integrate various data management functionalities to Python applications that you will develop while going through this course.

Further, after going through the content of this block you will learn the role of Python in mobile application development. Firstly, you will get an insight to the existing mobile application development environments. Then you will learn the suitable development environments and different open source Python libraries available to support these development environments.

Furthermore, this block will list the libraries that you can use in Android application development using Python. Kivy is one of the frameworks used by Python programmers to develop mobile applications. At the latter part of this unit, the steps to follow in setting up the application development environment will be explained.

Later, by understanding the content covered in this block you will learn the development of GUI components in Python and Tkinter standard GUI library that is bundled with python and wxPython libraries. You will also learn different types of packages available for GUI development.

Developing GUI using kivy is different from Tkinter and wxPython because Kivy has a different architecture. You will learn how Kivy GUI libraries can be used in python programming by studying the given programming codes. Output of each programming code is given before the code snippet. Write and run codes and check whether the outputs are correct.

Python with kivy cross platform is a very useful method of implementing GUI for mobile application. Four different types of GUI programming examples are given to study different areas.

This block consists of 5 units and is organized as follows:

Unit-11 Introduces the concept of Handling Data with Python

Unit – 12 Makes you to understand the Role of Python in Mobile application Development

Unit-13 Introduces the Mobile application development with python

Unit -14 Relates to Python graphical User Interface development

Unit – 15 Introduces GUI programming using KIVY Libraries

*Happy Programming !!*





---

# UNIT 11 HANDLING DATA WITH PYTHON

---

- 11.1 Introduction
- 11.2 Objectives
- 11.3 Terminologies
- 11.4 What is a Database?
- 11.5 Database Concepts
- 11.6 Introduction to SQLite
  - 11.6.1 Installing SQLite manager on Firefox
  - 11.6.2 Creating a database Connection
- 11.7 SQL CRUD statements
- 11.8 Introduction to database constraints
  - Video 13: Handling Data in Python
- 11.9 Summary
- 11.10 Further Reading

---

## 11.1 INTRODUCTION

---

In this unit you will learn about the importance of databases in real world applications and importance of using SQLite with Python.

First, you will learn about integrating database concepts into real world applications and data management using SQLite. Next, you will learn how to use these different techniques and when to use them.

It is more complicated to write the code to use a database to store data than Python dictionaries or flat files so there is little reason to use a database unless your application truly needs the capabilities of a database.

The situations where a database can be quite useful are:

- 1) When your application needs to make many small random updates within a large data set,
- 2) When your data is so large that it cannot fit in a dictionary and you need to look up information repeatedly, or
- 3) You have a long-running process that you want to be able to stop and restart and retain the data from one run to the next.

You can build a simple database with a single table to suit many application needs, but most problems will require several tables and links/relationships between rows in different tables. When you start making links between tables, it is important to do some thoughtful design and follow the rules of database normalization to make the best use of the database's capabilities. Since the primary motivation for using a database is that you have a large amount of data to deal with, it is important to model your data efficiently so your programs run as fast as possible.



On completion of this unit, you will be able to integrate these data management functionalities to Python applications that you develop.

---

## 11.2 OBJECTIVES

---

Upon completion of this unit you will be able to:

- *Setup* database designing environment for SQLite.
- *Identify* the constraints when designing a database.
- *Create* a database using SQLite and connect with a Python application.
- *Use* appropriate CRUD operations to manipulate data.

---

## 11.3 TERMINOLOGIES

---

<b>DBMS:</b>	A DataBase Management System (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data.
<b>CRUD:</b>	create, read, update, and delete (as an acronym CRUD) are the four basic functions of persistent storage.
<b>SQL Constraint:</b>	SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table.

---

## 11.4 WHAT IS A DATABASE?

---

A database is a collection of 'well-organised' information, so that it can be easily accessed, managed and updated. Inside a database, data is organized into rows, columns and tables (like a spreadsheet), and it is indexed to make it easier to find relevant information. Most of the e-commerce sites and other types of dynamic websites are experiencing this flexible feature in their applications.

Normally, a database management system provides users with the ability to control, read and/or write data in a database. Data get updated, expanded and deleted as new data are added. Databases are capable of creating and updating themselves, querying the data that they contain and running applications against them. Normally, a database manager provides users with the ability to perform the above tasks.

There are many different database management systems (DBMS) including: Oracle, MySQL, Microsoft SQL Server, and SQLite which are used for a wide variety of purposes.

We will focus on SQLite DBMS in this session because, it is a very common database and is already built into Python. SQLite is designed to be embedded into other applications to provide database support within the application.

The Firefox browser also uses the SQLite database internally as do many other products.

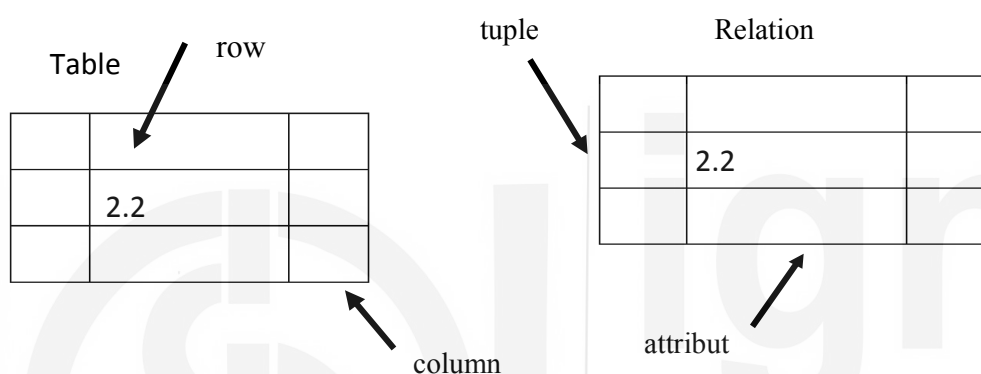
You will learn more about integrating database systems into applications in a later unit. In the following section, we will be discussing about the basic idea behind the database concepts.

---

## 11.5 DATABASE CONCEPTS

---

As we discussed before, when you first look at a database it will look like a spreadsheet with multiple sheets. The primary data structures in a database are: tables, rows, and columns. In technical terms, the concepts of table, row, and column can be more formally referred to as relation, tuple, and attribute, respectively.



Now, let us get familiar with these new terms.

- Relation: An area within a database that contains tuples and attributes. More typically called a “table”.
- Tuple: A single entry in a database table that is a set of attributes. More typically called “row”.
- Attribute: One of the values within a tuple. More commonly called a “column” or “field”.

---

## 11.6 INTRODUCTION TO SQLITE

---

SQLite is an embedded and easy to use relational database engine. It is a self-contained, server-less, zero-configuration and transactional SQL database engine. It is very fast and lightweight, and the entire database is stored in a single disk file. SQLite is used in many applications as internal data storage. The Python Standard Library includes a module called "sqlite3" intended for working with this database. This module is a SQL interface compliant with the DB-API 2.0 specification.

### 11.6.1 Installing SQLite manager on Firefox

Since this unit focuses on developing mobile applications using Python to work with data in SQLite database files, many operations can be done more

conveniently using a Firefox add-on called the SQLite Database Manager, which is freely available on:

<https://addons.mozilla.org/en-us/firefox/addon/sqlite-manager/>

Using the browser you can easily create tables, insert data, edit data, or run simple SQL queries on data in the database.

### 11.6.2 Creating a database Connection

To use the SQLite3 module we need to add an *import* statement to our Python script:

```
import sqlite3
```

When we create a database table we must tell the database that the names of each column and the type of data which we are planning to store in each column.

Various data types supported by SQLite are depict in below table:

Data types
VARCHAR(10),
NVARCHAR(15)
TEXT
INTEGER
FLOAT
BOOLEAN
CLOB
BLOB
TIMESTAMP
NUMERIC(10,5)
VARYING CHARACTER (24)
NATIONAL VARYING CHARACTER(16)

The code to create a database file and a table named 'StudentDetails' with two columns in the database is shown in Example 11.1

#### Example 11.1

```
import sqlite3

conn = sqlite3.connect('school.sqlite3')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS StudentDetails')
cur.execute('CREATE TABLE StudentDetails (studentName
TEXT, StudentID INTEGER)')
conn.commit()
conn.close()
```

The connect operation makes a “connection” to the database stored in the file `school.sqlite3` in the current directory. If the file does not exist, it will be created. The reason this is called a “connection” is that, sometimes the database is stored in a separate “database server” than the server on which we are running our application. In our simple examples, the database will just be a local file in the same directory as the Python code we are running.

A cursor is like a file handler that we can use to perform operations on the data stored in the database. Calling `cursor ()` is very similar conceptually to calling `open()` when dealing with text files.

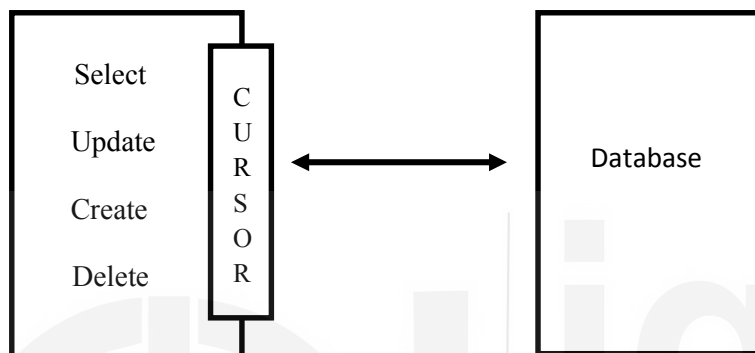


Figure 11.1 Available SQLite commands

Once we have the cursor, we can begin to execute commands on the contents of the database using the `execute()` method.

Database commands are expressed in a special language that has been standardized across many different database vendors to allow us to learn a single database language. That is called Structured Query Language or SQL for short.

The first SQL command removes the `StudentDetails` table from the database if it exists.

```
cur.execute('DROP TABLE IF EXISTS StudentDetails')
```

Then the second command will create a table named `'StudentDetails'` with a text column named `'StudentName'` and an integer column named `'StudentID'`.

```
cur.execute('CREATE TABLE StudentDetails (StudentName
TEXT, StudentID INTEGER)')
```

## 11.7 SQL CRUD STATEMENTS

### Insert Command

Now that we have created a table named `StudentDetails`, we can insert or enter some data into that table using the SQL `INSERT` operation. The SQL

INSERT command indicates which tables we are using and then defines a new row by listing the fields we want to include (StudentName, StudentID) followed by the VALUES (we specify the values as question marks (?, ?)) and a list of corresponding values for each of the fields.

### Example 11.2

```
cur.execute('INSERT INTO StudentDetails (StudentName,  
StudentID) VALUES ( ?, ? )', ( 'ABC Perera ', 15033 )  
)  
conn.commit()
```

### Select Command

Then we use 'SELECT' command to retrieve rows and columns from a database table. The SELECT statement lets you specify which columns you would like to retrieve as well as a WHERE clause to select which rows you would like to see. It also allows an optional ORDER BY clause to control the sorting of the returned rows.

```
cur.execute(SELECT * FROM StudentDetails WHERE  
StudentID = 3513)  
conn.commit()
```

Using \* indicates that you want the database to return all of the columns for each row that matches the WHERE clause.

You can also request that the returned rows be sorted by one of the fields as shown in Example 11.3.

### Example 11.3

```
cur.execute (SELECT StudentName, StudentID FROM  
StudentDetails ORDER BY StudentID)  
conn.commit()
```

### Delete Command

To remove a row, you need to use 'WHERE' clause on a 'DELETE' statement. The WHERE clause determines which rows are to be deleted:

### Example 11.4

```
cur.execute(DELETE FROM StudentDetails WHERE StudentID  
= 15033)  
conn.commit()
```

## Update Command

The UPDATE statement specifies a table and then a list of fields and values to change after the SET keyword and then an optional WHERE clause to select the rows that are to be updated. If a WHERE clause is not specified, it performs the UPDATE on all of the rows in the table.

It is possible to UPDATE a column or columns within one or more rows in a table using 'UPDATE' statement as in Example 11.5:

### Example 11.5

```
cur.execute(UPDATE StudentDetails SET StudentName =  
'XYZ Perera' WHERE StudentID = 15033)  
conn.commit()
```

## Alter table Command

The ALTER statement specifies a table name and the database which contains that table along with the 'RENAME TO' with a list of fields and values to change after the ALTER keyword.

It is possible to MODIFY a table name in a database using 'ALTER' statement as in 11.6:

### Example 11.6

```
cur.execute(ALTER TABLE school.StudentDetails RENAME  
TO school.StudentInformaiton)  
conn.commit()
```

## Drop table Command

The DROP TABLE statement specifies a database name after the DROP TABLE keyword

It is possible to DELTE a database using ' DROP' statement as shown in Example 11.7:

### Example 11.7

```
cur.execute(DROP TABLE school.StudentInformaiton)  
conn.commit()
```

These four basic SQL commands (INSERT, SELECT, UPDATE, and DELETE) allow the four basic operations needed to create and maintain data.

---

## 11.8 INTRODUCTION TO DATABASE CONSTRAINTS

---

When we design our table structures, we can tell the database that we would like it to enforce few rules on it. These rules will prevent us from making

mistakes and ensure the accuracy of the data that we insert. There are four major constraints named as

- Key Constraint
- Domain Constraint
- Entity Integrity Constraint
- Referential Integrity Constraint

### Example 11.8

```
cur.execute('''CREATE TABLE IF NOT EXISTS People
              (id INTEGER PRIMARY KEY, nic TEXT INTEGER, retrieved
              INTEGER)''')

cur.execute('''CREATE TABLE IF NOT EXISTS Follows
              (from_id INTEGER, to_id INTEGER, UNIQUE(from_id,
              to_id))''')
```

We indicate that the People table must be UNIQUE. We also indicate that the combination of the two numbers in each row of the Follows table must be unique. These constraints keep us from making mistakes such as adding the same relationship more than once.

We can take the advantage of these constraints in the following code:

### Example 11.9

```
cur.execute('''INSERT OR IGNORE INTO People (name,
retrieved)
VALUES ( ?, 0)''', ( friend, ) )
```

We add the OR IGNORE clause to our INSERT statement to indicate that if this particular INSERT would cause a violation of the “name must be unique” rule, the database system is allowed to ignore the INSERT. We are using the database constraint as a safety net to make sure that we do not inadvertently do something incorrect.

Similarly, the following code ensures that we don’t add the exact same Follows relationship twice.

```
cur.execute('''INSERT OR IGNORE INTO Follows
              (from_id, to_id) VALUES (?, ?)''', (id, friend_id) )
```

Again, we simply tell the database to ignore our attempted INSERT if it would violate the uniqueness constraint that we specified for the Follows rows.

### Key Constraints

Now that we have started building a data model putting our data into multiple linked tables, and linking the rows in those tables using keys, we need to look at some terminology around keys. There are generally three kinds of keys used in a database model.

A **logicalkey** is a key that the “real world” might use to look up a row. In our example data model, the name field is a logical key. It is the screen name for the user and we indeed look up a user’s row several times in the program using the name field. You will often find that it makes sense to add a UNIQUE constraint to a logical key. Since the logical key is how we look up a row from the outside world, it makes little sense to allow multiple rows with the same value in the table.

A **primarykey** is usually a number that is assigned automatically by the database. When we want to look up a row in a table, usually searching for the row using the primary key is the fastest way to find the row. Since primary keys are integer numbers, they take up very little storage and can be compared or sorted very quickly.

A **foreignkey** is usually a number that points to the primary key of an associated row in a different table. An example of a foreign key in our data model is the `from_id`.

We are using a naming convention of always calling the primary key field name `id` and appending the suffix `_id` to any field name that is a foreign key.

### Video 13: Handling Data in Python

This video will further explain data manipulation in Python. You may watch this Video with screencast an attempt Activity11.1

URL : <https://youtu.be/gEMPzwcSfC4>



---

#### Activity11.1

- Create a database using SQLite called ‘ABC\_Organization’
- Create a table inside that database and name it ‘Employee’,
- Create following fields in ‘Employee’ table

Name of the Field	Data type
EmployeeID	Integer (Primary Key)
EmpFirstName	Text
EmpLastName	Text
Gender	Boolean
NICNo	varchar(100)



- Insert an employee detail to the table (ex: EmployeeID : 105524, EmpFirstName : PQR, EmpLastName : Fernando, Gender : 1, NICNo : 895562987V)
  - Update the first name of the employee to 'PQWR' where employee id is 105524
  - Select fields EmpFirstName, EmpLastName as EmployeeName and NICNo from Employee table
- 

### Check Your Progress

Q-1 Write SQL Crud statements with example for each.

Q-2 What are integrity constraints? Briefly discuss various types of integrity constraints, Give example of each.

---

## 11.9 SUMMARY

---

This unit covered a lot of ground to give you an overview of the basics of using a database in Python it also provide few examples to create database connection and to write SQL commands. Further this unit discussed how to integrate data management functionalities to Python applications.

---

### 11.10 FURTHER READING

---

- 1) Eng, N., & Watt, A. (2013). Database design. Retrieved July 08, 2016, from BC Open TextBooks, <https://opentextbc.ca/dbdesign/chapter/chapter-3-characteristics-and-benefits-of-a-database/>
  - 2) Andres Torres, Python. "Introduction To Sqlite In Python | Python Central". Python Central. N.p., 2017. Retrieved 25 January 2017 from
  - 3) Bodnar, Jan. "Introduction To Sqlite - Sqlite Description And Definitions". Zetcode.com. N.p., 2017. Retrieved 28 January 2017 from
- 

### 11.10 ANSWER TO CHECK YOUR PROGRESS

---

Ans-1 Refer section 11.7

Ans-2 Refer section 11.8

---

## UNIT 12 ROLE OF PYTHON IN MOBILE APPLICATION DEVELOPMENT

---

- 12.1 Introduction
- 12.2 Objectives
- 12.3 Terminologies
- 12.4 Mobile Application Development Environments
- 12.5 Uses of Python in Mobile Application Development
- 12.6 Open Source Python Libraries
- 12.7 Kivy
- 12.8 Summary
- 12.9 Further Reading
- 12.10 Answer to check your progress

---

### 12.1 INTRODUCTION

---

In this unit we are going to learn the role of Python in mobile application development. First part of this unit will give you an insight to the existing mobile application development environments. Then you will learn the suitable development environments and different open source Python libraries available to support these development environments.

Furthermore, this unit will list the libraries that you can use in Android application development using Python. Kivy is one of the frameworks used by Python programmers to develop mobile applications. At the latter part of this unit, the steps to follow in setting up the application development environment will be explained.

---

### 12.2 OBJECTIVES

---

Upon completion of this unit you will be able to:

- *identify* different mobile application development environments
- *explain* the use of open source Python libraries for rapid development of applications
- *describe* the use of python libraries available for android application development
- *illustrate* the Kivy app architecture using a diagram
- *set up* the application development environment using Kivy

---

## 12.3 TERMINOLOGIES

---

- Kivy:** an open source Python library for developing mobile apps and other multitouch application software
- pip:** pip is a package management system used to install and manage software packages written in Python.
- wheel:** A built-package format for Python
- OpenGL:** Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics

---

## 12.4 MOBILE APPLICATION DEVELOPMENT ENVIRONMENTS

---

There is a variety of applications in various fields including business, entertainment, utilities, hospitality sector, games and much more, whose apps are made sure to fit to various screen sizes be it a smart phone, a tab or any other device. Android, iOS and Windows are few of the operating systems that run on these devices. Different development platforms are being used when developing mobile applications to run on top of these operating systems. The following table lists some of such development platforms used by developers all over the world.

**Table 12.1: List of mobile application development platforms**

Mobile Operating System	Mobile Application Development Platform	Programming Language
Android	Android Studio Android SDK, Eclipse	Java
iOS	Xcode IDE	Objective C
Windows	Visual Studio	C#

All the platforms listed above allow developers to implement mobile applications to run on Android, iOS or Windows only. Mobile application developers try to develop applications, making sure to fit to various screen sizes and operating systems.

Cross platform mobile development essentially makes use of frameworks allowing developers to create platform independent mobile applications predominantly utilising already familiar web standards like HTML/JavaScript/CSS. These frameworks act more or less as a middleware or bridge and provide the platform specific implementation of API in the native

programming language for the language of the framework to communicate with the native code of different platforms.

The top tools used for cross-formatting mobile application development are RhoMobile, PhoneGap, Appcelerator, Sencha Touch, MoSync, Whoop, WidgetPad, GENWI, AppMakr, Mippin, SwebApps, MobiCart, etc.

---

## 12.5 USES OF PYTHON IN MOBILE APPLICATION DEVELOPMENT

---

As you have already learnt, Python is not widely used in mobile application development. Still Python is used mostly in cross platform development due to its platform independent nature. Python runs on all major operating systems such as Windows, Linux/Unix, OS/2, Mac, Amiga, etc. The uses of Python in mobile application development are given below.

- To write mobile applications to run on multiple platforms
- As a scripting language to run on mobile devices

Android Google provides Android Scripting Environment (ASE) which allows scripting languages including Python to run on Android. QPython is another script engine that also runs on android devices like phone or tablet. It lets your android device run Python scripts and projects.

An example of using Python as a scripting language in Android is explained in this section.

Batterystats is part of the Android framework and collects battery data from any Android device. Battery Historian is an open-sourced project which is available on GitHub, which converts the data collected from Batterystats into an HTML visualisation that can be viewed in a browser.

To work with Batterystats and Battery Historian a Python script can be used and the steps to be followed are given below.

Step 1: Download the open-source Battery Historion Python script from GitHub (<https://github.com/google/battery-historian>).

Step 2: Unzip the file to extract the Battery Historian folder. Inside the folder, find the **historian.py** file and move it to the Desktop or another writable directory.

Step 3: Connect your mobile device to your computer.

Step 4: On your computer, open a Terminal window in Android Studio.

Step 5: Change to the directory where you've saved **historian.py**, for example: `cd ~/Desktop`

Step 6: Shut down your running adb server by entering the following command. `> adb kill-server`

Step 7: Restart adb and check for connected devices by entering the following command. You will see a list of devices attached. `> adb devices`

If a list of devices is not seen, then may be your phone is not connected properly. So, connect the phone and turn on USB Debugging. Then you should kill and restart adb.

Step 8: Reset battery data gathering by entering the following command. > adb shell dumpsys batterystats --reset

When you reset, old battery collection data will get erased. Otherwise, there will be huge output.

Step 9: Disconnect your device from the computer to make sure that it draws current only from the device's battery.

Step 10: Use the particular app for a short time.

Step 11: Connect your phone again

Step 12: See whether your phone is recognised (use > adb devices)

Step 13: Then dump all battery data using the following command. This action may take some time. > adb shell dumpsys batterystats > batterystats.txt

Step 14: Create a HTML version of the data dump for Battery Historian: > Python historian.py batterystats.txt > batterystats.html

Step 15: Open the batterystats.html file in your browser.

You can open the `historian.py` file and study how the Python script has been written.

Source: <https://developer.android.com/studio/profile/battery-historian.html>

---

## 12.6 OPEN SOURCE PYTHON LIBRARIES

---

The language you choose for mobile development varies depending on many factors. Other than Java, C# and Objective C there are many more languages that support mobile development. Some of the examples are HTML5 for front end, C++ for Android and Windows development, Swift to work along with Objective C. In this section we will look at the open source Python libraries available for rapid development of applications.

- Kivy - Kivy is a multi-platform application development kit, using Python
- PyGame - PyGame is a set of Python modules designed for writing games. PyGame allows us to easily program games in Python and port them to an Android application.
- PGS4A - Pygame Subset for Android
- SL4A- The SL4A project makes scripting on Android possible, it supports many programming languages including Python, Perl, Lua, BeanShell, JavaScript, JRuby and shell.
- PyGTK – PyGTK allows to create programs with a graphical user interface using the Python programming language

- **WXPython** - WXPython is a GUI toolkit for Python programming language. This is implemented as a Python extension module (native code) that wraps the wxWidgets cross platform GUI library, which is written in C++.
- **PyQT and PySide** – These are the two popular Python bindings for the Qt cross-platform GUI/XML/SQL C++ framework. Qt is a cross-platform application framework that is used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with native capabilities and speed.
- **QPython** - QPython is a script engine which runs Python programs on android devices.
- **VPython** - VPython allows users to create objects such as spheres and cones in 3D space and displays these objects in a window
- **TkInter** - TkInter is Python's standard GUI (Graphical User Interface) package.

As you can see there are many open source libraries to develop different types of applications for multiple platforms. Kivy is one of such platform facilitate application development for multiple platforms. We will learn about Kivy in detail in the next section.

---

## **12.7 KIVY**

---

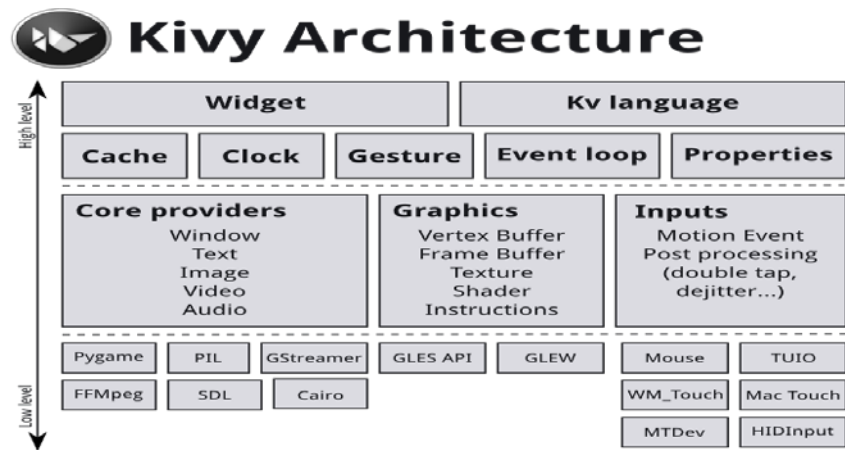
Kivy allows you to write your code once and have it run on different platforms. This section will provide you a guide to get the tools you need, understand the major concepts and learn best practices. As this is an introduction, pointers to more information in developing an application will be given in Unit 13.

Using Kivy on your computer, you can create applications that run on:

- Desktop computers: OS X, Linux, Windows.
- iOS devices: iPad, iPhone.
- Android devices: tablets, phones.
- Any other touch-enabled devices supporting TUIO (Tangible User Interface Objects)

### **Kivy Architecture**

Let us look at the architectural view of Kivy. Knowing the Kivy architecture will help you when developing applications using Kivy platform.



**Figure 12.1: Kivy Architecture**

Source: <https://kivy.org/docs/guide/architecture.html>

Let's briefly look at the details of following components.

- **Core Providers:** Core providers are the abstractions of basic tasks or core tasks such as opening a window, displaying text and images, playing audio, getting images from a camera, correcting spelling etc.
- **Input Providers:** An input provider is a program that facilitate for a specific input device. When a new input device is added, you need to provide a new class that reads the input data from the new device and transforms them into basic events.
- **Graphics:** Graphics API of Kivy is an abstraction of Open Graphics Library (OpenGL). Within the the software, Kivy issues hardware-accelerated drawing instructions using OpenGL.
- **Core:** The programs in the core package provides commonly used features, such as calendar, clock, cache, gesture detection, Kivy language and properties. Properties link your widget code with the user interface description. (Kivy language is used to describe user interfaces)
- **UIX (Widgets & Layouts):** It is the UIX module that contain commonly used widgets and layouts. These can be re-used to create a user interface quickly. You will learn how to import Labels from widgets in the next section while writing your first program in Python.
- **Modules:** Modules are used to add extra functions into Kivy programs
- **Input Events (by touch):** Kivy abstracts different input types and sources such as touch screens, mice, and any other Tangible User Interface Objects (TUIO).

### Installation of the Kivy environment

To use Kivy you need to install Python first. You may have multiple versions of Python installed side by side, then you have to install Kivy for each version of Python.

- 1) Before installing Kivy you need to ensure you have the latest pip and wheel. wheel is a packaging format used. For installing them you need to use the following command.

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

2) Next you need to install the dependencies.

wheels are available for dependencies separately so only necessary dependencies need to be installed. The dependencies are offered as optional sub packages of kivy.deps, e.g. kivy.deps.sdl2

Currently on Windows, the following dependency wheels are given:

- gstreamer for audio and video
- glew for OpenGL, if you are using Python 3.5 angle can be used instead of glew
- sdl2 for control and/or OpenGL

To install the dependencies you need to use the following command:

```
python -m pip install docutils pygments pypiwin32
kivy.deps.sdl2 kivy.deps.glew
python -m pip install kivy.deps.gstreamer
```

3) Once the dependencies are installed, the environment is ready to install Kivy.

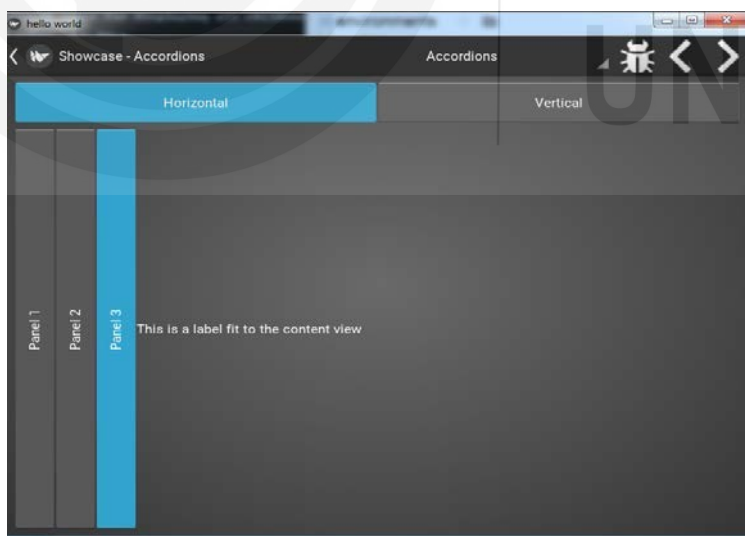
To install Kivy the following command to be used.

```
python -m pip install kivy
```

4) Now we can import kivy to python or run a basic example.

Let us us a sample Python program given in kivy-examples now.

```
python share\kivy-examples\demo\showcase\main.py
```



**Figure 12.2 : Output of the main.py program**

You will require a basic knowledge of Python to start developing applications using Kivy.

### **Create an application using Kivy**

Creating a kivy application is simple if you are familiar with Python and know how to apply object oriented concepts.



An example of a minimal application is given below.

```
import kivy
kivy.require('1.0.6')
# replace with your current kivy version !

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

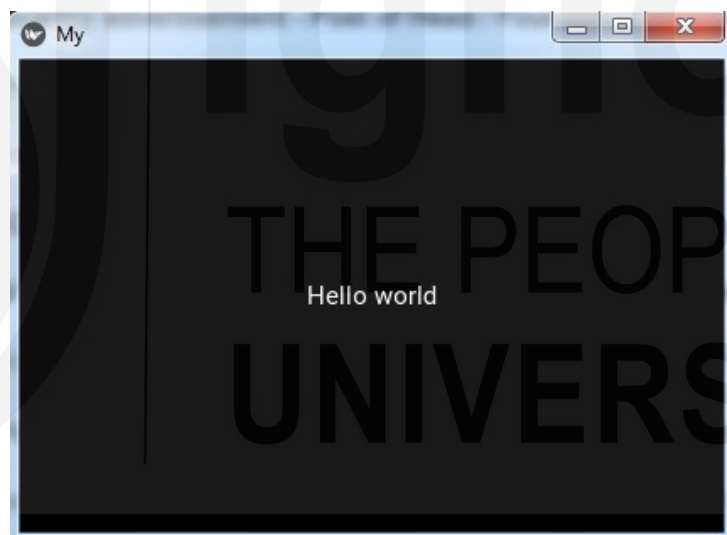
class MyApp(App):
    def build(self):
        return Label(text='Hello world')
if __name__ == '__main__':
    MyApp().run()
```

You can save this to a file, `main.py` for example, and run it.

If you saved your file inside the Python installation folder, you need to use the following command to run the program.

```
python main.py
```

You will find the output of this program as given below.



**Figure 12.3: Output of `first.py`**

- In order to use Kivy, it's required to `import kivy` first; line 1 shows how to `import kivy`.
- The term `require` can be used to check the minimum version required to run a Kivy application. To run this program you need to have '1.0.6'
- Line 3 is required so that the base class of your `App` inherits from the `App` class. It's present in the `kivy_installation_dir/kivy/app.py`
- In line 4, the `uix` module is the section that holds the user interface elements like layouts and widgets.

The above program has the following three parts.

- In line 5, sub-classing the `App` class

This is where we are *defining* the Base Class of our Kivy App. You should only need to change the name of your app `MyApp` in this line.

- In line 6, implementing its `build()` method so it returns a Widget instance. The Label widget is for rendering text. It supports ASCII and unicode strings.

Here we initialize a Label with text 'Hello World' and return its instance. This Label will be the Root Widget of this App.

- In line 9, instantiating this class, and calling its `run()` method

Let us look at the source code for another simple application created to draw circles and lines as shown in the following figure.



Figure 12.4: Output of `paint.py`

Source extracted from Kivy documentation in <https://kivy.org/docs/tutorials/firstwidget.html>

- Here in Line 1, we import Python's `random()` function that will give us random values in the range. `import` statement given as line 4 is required to use the `Button` class.
- In line 8, we create a new tuple of 3 random float values that will represent a random RGB color. Since we do this in `on_touch_down`, every new touch will get its own colour.

- In line 10 we set the color for the canvas. We use the random values we

```
from random import random #Line1

from kivy.app import App

from kivy.uix.widget import Widget #Line4
from kivy.graphics import Color, Ellipse, Line

class MyPaintWidget(Widget):
    def on_touch_down(self, touch):
        color = (random(), 1, 1) #Line8
        with self.canvas:
            Color(*color, mode='hsv') #Line10
            d = 30.
            Ellipse(pos=(touch.x-d/2, touch.y-d/2), size=(d, d))
            touch.ud['line'] = Line(points=(touch.x, touch.y))

    def on_touch_move(self, touch):
        touch.ud['line'].points += [touch.x, touch.y]

class MyPaintApp(App):
    def build(self):
        parent = Widget() #Line 18

        self.painter = MyPaintWidget() #Line 19

        clearbtn = Button(text='Clear') #Line 20
        clearbtn.bind(on_release=self.clear_canvas) #Line 21
        parent.add_widget(self.painter) #Line 22

        parent.add_widget(clearbtn) #Line 23
        return parent

    def clear_canvas(self, obj): #Line 25
        self.painter.canvas.clear() #Line 26

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

generated only at this time and feed them to the colour class using Python's tuple unpacking syntax.

- In Line 18, `parent = Widget()` is used to create a
- dummy `Widget()` object as a parent for both our painting widget and the button we're about to add.
- In line 19, we create our `MyPaintWidget()` as usual, only this time we don't return it directly but bind it to a variable name.
- In line 20, We create a button widget. It will have a label on it that displays the text 'Clear'.

In line 21, we bind the button's `on_release` event (which is fired when the button is pressed and then released) to the callback

function `clear_canvas` defined on below on lines 25 & 26.

We set up the widget hierarchy in line 22 and 23 by making both the painter and the `clearbtn` children of the dummy parent widget. That means painter and `clearbtn` are now siblings in the usual computer science tree terminology.

Up to now, the button did nothing. It was there, visible, and you could press it, but nothing would happen. We change that in lines 25 and 26. We create a small, throw-away function that is going to be our callback function when the button is pressed. The function just clears the painter's canvas' contents, making it black again.

---

### Activity 12.1

- List down and briefly explain the use of five open source Python libraries which are not mentioned in the unit.

---

### Check Your Progress

- Q-1 What is the utility of Battery stats and Battery Historian tools? How one can use python scripts to work with Battery stats and Battery Historian? Discuss.
- Q-2 draw the architecture of Kivy and discuss the functionality of various components involved in the architecture.
- Q-3 Create an application in Kivy to display “You are welcome”

---

## 12.8 SUMMARY

Different types of Python libraries available for rapid application development were explained in this unit. Furthermore the details of Kivy application life cycle and the Kivy architecture were explained. Steps to be followed when writing a Python application using Kivy were explained with examples at the end of the unit.

In the next section you will learn how the developed applications using Kivy can be packaged to run on Android devices.

---

## 12.9 FURTHER READING

Kivy is an Open Source Python Library. <https://kivy.org/#home>  
<https://kivy.org/doc/stable/guide/architecture.html>

Contributors for Kivy (as given on Kivy.org by March 2019)

- Terje Skjaeveland (bionoid)
- George Sebastian (georgs)
- Gabriel Ortega
- Arnaud Wael (triselectif)
- Thomas Hirsch
- Joakim Gebart
- Rosemary Sebastian
- Jonathan Schemoul

Past core developers

- Thomas Hansen (hansent)
- Christopher Denter (dennda)
- Edwin Marshall (aspidites)
- Jeff Pittman (geojeff)
- Brian Knapp (knappador)
- Ryan Pessa (kived)

- Ben Rousch (brousch)

Special thanks

- Mark Hembrow
- [Vincent Autin](#)

---

## 12.10 ANSWER TO CHECK YOUR PROGRESS

---

Ans-1 Refer section 12.5

Ans-2 Refer section 12.7

Ans-3 Refer section 12.7



---

## UNIT 13 MOBILE APPLICATION DEVELOPMENT WITH PYTHON

---

- 13.1 Introduction
- 13.2 Objectives
- 13.3 Terminologies
- 13.4 Android Mobile Application Development using Kivy
- 13.5 Buildozer Tool
- 13.6 Packaging with Python-for-android
- 13.7 Packaging your application for the Kivy Launcher
- 13.8 The Kivy Android Virtual Machine
- 13.9 Summary
- 13.10 Further Reading
- 13.11 Answer to check your progress

---

### 13.1 INTRODUCTION

---

In the previous unit you learnt to set up the development environment for a Kivy application. In this unit you will be learning to develop an application using Kivy and to package it to run on Android devices.

---

### 13.2 OBJECTIVES

---

Upon completion of this unit you will be able to:

- *Develop* a Python application using Kivy
- *Build* the developed application and run it on Android device

---

### 13.3 TERMINOLOGIES

---

**bootstrap:** A bootstrap is a class consisting of few basic components (i.e. with SDL2, Pygame, Webview etc.)

**Virtual Machine(VM):** A virtual machine is an operating system or application environment that is installed on software.

---

### 13.4 ANDROID MOBILE APPLICATION DEVELOPMENT USING KIVY

---

A Kivy application can run on Android device. For that we have to compile a Kivy application and to create an Android APK which will run on the device similar to a Java application. It is possible to use different tools which will help to run code on Android devices. We can publish these APKs to Google store where users can download and install in their devices or it is possible to

run the apps using a Kivy Launcher app. These two methods will be explained further in this section.

- First method is to use the prebuilt Kivy Android VM image, or use the Buildozer tool to automate the entire process.
- The second method is to run the Kivy app without a compilation step with the Kivy Launcher app

---

## 13.5 BUILDZOER TOOL

---

Buildozer is a tool that allows packing of mobiles application easily. It downloads and sets up all the prerequisites for Python-for-android, including the android SDK and NDK, then builds an apk that can be automatically pushed to the device.

Buildozer currently works only in Linux. The steps to follow when using Buildozer on Linux are given below.

You can get a clone of Buildozer from <https://github.com/kivy/buildozer> For that you need to use the following commands in Linux

```
Git clone https://github.com/kivy/buildozer.gitcd buildozer sudo python2.7  
setup.py install
```

This will install Buildozer in your system. Afterwards, navigate to your project directory and run:

```
buildozer init
```

This creates a buildozer.spec file controlling your build configuration. You should edit it appropriately with your app name etc. You can set variables to control most or all of the parameters passed to Python for android.

You need to install buildozer's dependencies and then you need to plug in your android device and run the application using the following command. This command will build, push and automatically run the apk on your device.

```
buildozer android debug deploy run
```

---

## 13.6 PACKAGING WITH PYTHON-FOR-ANDROID

---

Python-for-android is also represented as P4A. To install P4A you need to use the following command. This section explains how to do this in Linux environment. The same process can be done in Windows on top of a Virtual Machine. This process is explained at the later part of this unit.

Python-for-android can be installed using the following command.

```
Python -m pip install python-for-android
```

Then you need to install the dependencies you need. Some of the dependencies are given below.

- git
- ant
- python2
- cython
- a Java JDK
- zlib
- libncurses unzip
- virtualenv ( can be installed via pip)
- ccache (optional)

You need to download and unpack the Android SDK and NDK to a directory (let's say \$HOME/Documents/)

Then, you can edit your ~/.bashrc or other favorite shell to include new environment variables necessary for building on android

```
# Adjust the paths!
export ANDROIDSDK="$HOME/Documents/android-sdk-21"
export ANDROIDNDK="$HOME/Documents/android-ndk-r10e"
export ANDROIDAPI="14" # Minimum API version your
application require
export ANDROIDNDKVER="r10e" # Version of the NDK you
installed
```

To build your application, you need to have a name, version, a package identifier, and explicitly write the bootstrap you want to use, as well as the requirements. For build options some of the existing bootstraps are i.e. with SDL2, Pygame, Webview etc.

To build an application you need to specify the information given above as shown below.

```
p4a apk --private $HOME/code/myapp -- package=org.example.myapp --name "My application" --
version
0.1 --bootstrap=sdl2 --requirements=python2,kivy
```

If anything goes wrong and you want to clean the downloads and builds to retry everything it is required to run the following command.

```
p4a clean_all
```

If you just want to clean the builds to avoid re-downloading dependencies you need to use the following command.

```
p4a clean_builds && p4a clean_dists
```

---

## 13.7 PACKAGING YOUR APPLICATION FOR THE KIVY LAUNCHER

---

The Kivy launcher can be used to run the Kivy applications on Android devices without compiling them. Kivy launcher runs the Kivy examples stored on the SD Card in the device. To install the Kivy launcher, you must



go to the Kivy Launcher page on the Google Play Store. Then click on Install and select your phone. If not you can go to <https://kivy.org/#download> and install the APK manually.

Once the Kivy launcher is installed, you can put your Kivy applications in the Kivy directory in your external storage directory. Often the application is available at /sdcard even in devices where this memory is internal. For an example /sdcard/kivy/<your application>

<your application> should be a directory containing your main application file(e.g. main.py) and a text file (android.txt) with the contents given below.

```
title= <Application Title>
author=<Your name>
orientation=<portrait|landscape>
```

---

## 13.8 THE KIVY ANDROID VIRTUAL MACHINE

---

So far you learnt to build a Kivy Android application in a Linux environment configured with Python-for-android, the Android SDK and the Android NDK. It is possible to use a fully configured VirtualBox disk image in Windows and OS X operating system, because using the previous methods were limited only for Linux based developers.

### Setting up the environment

Step 1 : Download the disc image from <https://kivy.org/#download>, in the Virtual Machine section. A Virtual Machine with Android SDK and NDK and all other pre-requisites pre installed to ease apk generation.

The size of the download is more than 2GB and around 6GB after extracted. Extract the file and remember the location of the extracted vdi file.

Step 2 : Download the version of VirtualBox for your machine from the VirtualBox download area and install it.

Step 3 : Start VirtualBox, click on “New” in the left top. Then select “linux” and “Ubuntu 64-bit”. You need to check the available Linux distribution and select appropriately.

Step 4 : Under “Hard drive”, choose “Use an existing virtual hard drive file”. Search for your vdi file and select it. Assigning insufficient memory may result in the compile failing with cryptic errors. Therefore it’s required to assign sufficient memory when creating the virtual machine.

Step 5 : Go to the “Settings” for your virtual machine. In the “Display -> Video” section, increase video ram to 32MB or above. Enable 3D acceleration to improve the user experience.

Step 6 : Start the Virtual machine and follow the instructions in the *readme* file on the desktop.

The instructions given in the *readme* file are given below for your reference. These instructions can be changed based on the virtualbox disk image that you are downloading.

To use it go into your project directory, and use:

```
buildozer init
```

you can then edit the created `buildozer.spec`, to suit your project. Then use

```
buildozer android debug
```

to build your project, once you have it built, you can use the

```
buildozer android deploy run logcat
```

command to start the app on your device and collect the error log. (use ctrl-c to stop the logcat) Commands can be combined as given below.

```
buildozer android debug deploy run logcat
```

This command will do the whole process.

To update Buildozer you need to use the following command.

```
sudo pip install -U buildozer
```

To update Kivy and other modules the simplest way is to remove the buildozer cache before building your distribution. You can do this by using the following command.

```
rm -rf ~/.buildozer/android/packages
```

### Building the APK

Once the VM is loaded, you can follow the instructions from Packaging with Python-for-android given above.

Generally, your development environment and toolset are set up on your host machine but the APK is build in your guest. VirtualBox has a feature called 'Shared folders' which allows your guest direct access to a folder on your host.

#### Activity13.1

- Download the Kivy demos for Android by visiting <https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/kivy/kivydemo-for-android.zip>
- Unzip the contents and go to the folder 'kivydemo-for-android'
- Copy all the subfolders here to /sdcard/kivy
- Run the Kivy launcher and select one of the Pictures, Showcase, Touchtracer, Cymunk or other demos

### Check Your Progress

Q-1 Compare Buildozer and Kivy Launcher.

## 13.9 SUMMARY

In this unit you learned how to package your Python application to run on Android devices. In the next section let's learn about how to design the Graphical User Interfaces(GUI) for mobile applications using Python libraries

## 13.10 FURTHER READING

Python for Android

<https://python-for-android.readthedocs.io/en/latest/quickstart/>

Create a package for Android

<http://kivy.readthedocs.io/en/latest/guide/packaging-android.html>

---

## 13.11 ANSWER TO CHECK YOUR PROGRESS

---

Ans-1 Refer section 13.4, 13.5 and 13.7



---

## UNIT 14 PYTHON GRAPHICAL USER INTERFACE DEVELOPMENT

---

- 14.1 Introduction
- 14.2 Objectives
- 14.3 Terminologies
- 14.4 Graphical User interface (GUI)
- 14.5 Different types of packages for GUI development in Python
- 14.6 Tkinter (GUI toolkit that comes with Python)  
Video 14: Creating GUI for Python with Tkinter
- 14.7 GUI with wxPython  
Video 15: Python GUI with WxFrame
- 14.8 Summary
- 14.9 Reference
- 14.10 Answer to check your progress

---

### 14.1 INTRODUCTION

---

In this unit you will learn components of GUI and Tkinter standard GUI library that is bundled with python and wxPython libraries. You will also learn different types of packages available for GUI development. There are eight (8) examples to study Tkinter standard library and three (3) examples to study wxPython.

Line numbers before the programming code are included for easy references. But do not to include line numbers when you code in the text editor. Essential lines in the code are described after the program code.

---

### 14.2 OBJECTIVES

---

Upon completion of this unit you will be able to:

- Describe Graphical User Interface (GUI) and its components.
- Identify different types of packages for python GUI implement.
- Demonstrate the skills of programming in Tkinter GUI standard library.
- Demonstrate GUI programming with wxPython cross platform libraries.
- Apply event driven programming for GUI using Tkinter and wxPython libraries

## 14.3 TERMINOLOGIES

- GUI:** Graphical User Interface
- widget:** building blocks that make up an application in a GUI
- Layouts:** layouts to arrange widgets
- import:** Import the necessary modules to python code
- Tkinter:** Standard python GUI library
- wxPython:** Free and open source GUI package for python

## 14.4 GRAPHICAL USER INTERFACE (GUI)

When you work with computer, you need to interact with it in many ways such as clicking on various icons, selecting from menus, right clicking, double clicking, writing text in word processor, inserting pictures, inserting movies etc. All these works are processed in computer through hardware. User Interface is the space where interactions between you and the computer hardware.

The first user interface that came with computers, was a command-line interface where you could interact with the computer by typing commands on the keyboard. Typing commands was a very difficult task and user needs to have a very good hardware background. Later, graphical representations of such commands were created to interact with any user which are called Graphical User Interface (GUI).

Visual indications of GUI which can be implemented by python language are illustrated in fig. 14.1.

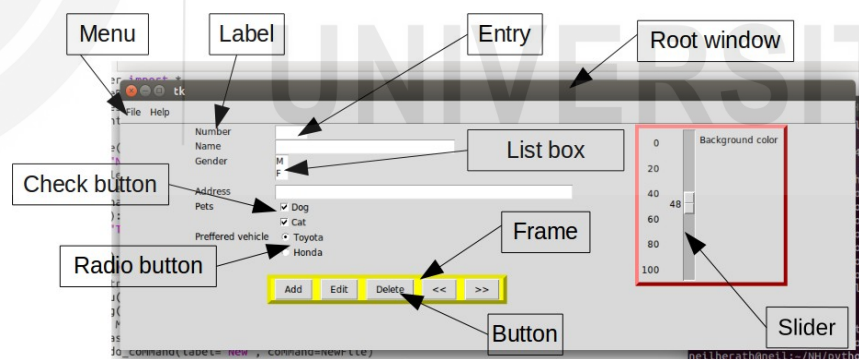


Figure 14.1 Visualization of GUI

Before we proceed, let's define some of the common terms.

**Window** – it is a rectangular zone on computer screen

**top-level window** - is an independent window within an application. One application may have many top level windows. Many children windows can be created within top level window. When you close a child window it will go back to the parent window

What is shown in figure 14.1 as the root window is a top level window.

**widget** - the general word for visual building blocks to interact an Activity in GUI.

**Layouts** - The way you arrange the widgets in a GUI.

**Frame** - Rectangular area which contains other widgets to organize different layouts.

**Parent, child** – A relationship of parent and child is created for any created widget. For example, when a check button placed on a frame, frame becomes *parent* and check button becomes *child*.

---

## 14.5 DIFFERENT TYPES OF PACKAGES FOR GUI DEVELOPMENT IN PYTHON

---

There are many tool kit options available for developing graphical user interface (GUI) . Most popular are given below:

**Tkinter** - GUI toolkit given with python is Tkinter. All the other GUI tool kits for python can be used as alternatives to the Tkinter.

**PyGTK** – permits to write GTK programs in python and has a lot of widegets than Tkinter.

**PyQT** – is a set of python software libraries for Qt Framework. Qt is an extensive C++ GUI application development framework that is available for Unix, Windows and Mac OS X.

**wxPython** – is a set of python software libraries to create wxWidgets.

**Kivy** – is an open source cross platform for developing mobile apps using python. Mobile apps can be developed for andriod, iOS, linux and windows.

GUI program development with Tkinter and wxPython is described next, whereas program development with Kivy GUI is described in unit 15.

---

## 14.6 TKINTER (GUI TOOLKIT THAT COMES WITH PYTHON)

---

GUI library for python given with the python programming language is Tkinter. GUI applications can be created easily and quickly using Tkinter. Strong object oriented widgets to the Tk GUI are given by Tkinter

### Creating GUI using Tkinter

Method of programming Tkinter GUI window is described by the example 14.1.

### Example 14.1 Tkinter GUI window

1	<code>#!/user/bin/python</code>
2	<code>import Tkinter as tk1</code>
3	<code>tp=tk1.Tk()</code>
4	<code># Codes for widgets</code>
5	<code>tp.mainloop()</code>

Window created by the above program is illustrated by Figure 14.2.

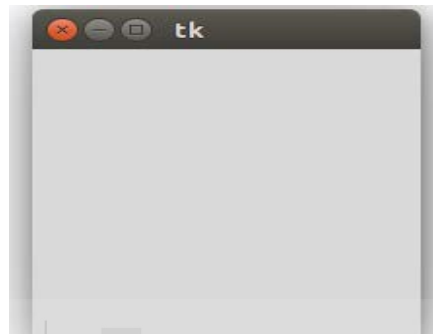


Figure 14.2. Tkinter window

Code line of the above program are explained as follows

1	Self executing script, in linux you can run by './' rather than typing python <file name>.py
2	Import Tkinter library
3	Tkinter Tk window is assigned to top.
4	Comment
5	Main loop is started and waiting for mouse and key board actoins.

### Video 14: Creating GUI for Python with Tkinter

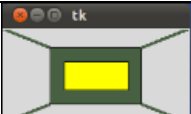
This video will demonstrate how to create a GUI with Tkinter. You may watch this Video carry out the tasks listed below in creating widgets, buttons etc.


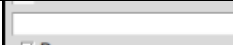

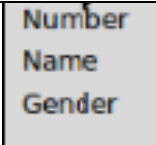



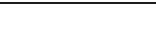
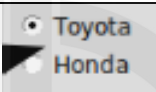


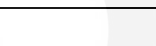



URL <https://youtu.be/jcgyfeZsaTc>



### Tkinter Widgets

Widget is a graphical user interface controls to interact between user and computer. Tkinter gives different types of controls such as buttons, labels and text boxes used in a GUI application. Tkinter widgets are described in the following table. Most widdegts can be seen in figure 14.1.

Widget	Widget image	Description
Canvas		is a rectangle area which used to display and edit graphics. For example – lines, polygen, rectangle

Checkbutton		Number of options can be selected using check buttons. More than one choice can be selected in grouped options
Entry		A line of text can be entered using entry widget
Frame		A container to insert and organize other widgets.
Label		Caption of single line can be displayed using label. Images may also insert to label.
Listbox		list of options can be provided listbox
Menubutton		Menu options can be displayed in an application.
Menu		Various commands contained in Menu buttons are given to user.
Message		For displaying multiline text fields to take values from user
Radiobutton		Only one option at a time can be selected by Radio button.
Scale		A number value can be changed by moving knob of a slider.
Scrollbar		Scrolling facility can be added for various widgets.
Text		Multiple line of text can be displayed.
Toplevel		Separate window container can be provided.
LabelFrame		Works as a container for complex window layouts.
tkMessageBox		Message box can be inserted to application.

### Using Button widget in window

Function or a method can be attached to a button which is called automatically when the button is clicked.

Here is the simple syntax to create this widget:

**w = Button ( master, option=value, ... )**

#### Parameters:

- master: Parent window
- options: Options are listed in the Following table.



Option	Description
Active background	Background color when the cursor is on the button.
Active foreground	Foreground color when the cursor is on the button.
Bd	width of the border. Default is 2.
Bg	Normal background color.
command	Calling function or method when the button is clicked.
Fg	Colour of the text (foreground)
Font	Type of the font used for label of the button.
Height	Height of the text or image
Highlight color	The color of the focus highlight when the widget has focus.
Image	Image to be attached and display
Justify	Alignment of the text line - justify
CENTER	Alignment of the text line - center
Padx	Horizontal length of button through x axis can be sat.
Pady	vertical length of button through x axis can be sat.
Relief	Border style- button appear as SUNKEN, RAISED, GROOVE, or RIDGE
State	Can set the button as DISABLED or ACTIVE. It active when the button is over.
underline	Underline the relevant character, Default is -1, which means no character is underlined.
Width	Set the width of the button.
wraplength	If this value is set to a positive number, the text lines will be wrapped to fit within this length.

**Example 14.2.** This python program illustrates how to work with button.

```

1  #!/usr/bin/python
2
3  import Tkinter as tk
4  import tkMessageBox as bx
5  tp=tk.Tk()
6  def exbutton():
7      bx.showinfo("Python button example","Hello
8  World")
9  B=tk.Button(tp,text="Press this
10 button",command=exbutton)
   B.pack()
   tp.mainloop()tp.mainloop()

```



Figure 14.3. Message “Hello World” will appear when "Press this button" is clicked

The output of the above program is shown in fig 14.3. When the button called ‘Press this button’ is clicked, the message box ‘Hello World’ will appear.

Description of codes in line numbers of the program

```
9. import tkinter module
10. a function called exbutton
11. Text of the button is "Press this button" and button is assigned to
    "B". Function exbutton is invoked by clicking button.
```

### Using Entry widget in window

Single line text strings can be entered using entry widget. Text widget can be used to enter multiple lines and label widget is used to display one or more lines of text but cannot edit.

### Syntax

Here is the simple syntax to create this widget –

```
w = Entry( master, option, ... )
```

### Parameters:

- **master:** the parent window.
- **options:** : List of ususalys used options are given below

Option	Description
Bg	background color
Bd	Size of the border The size of the border around the indicator. Default is 2 pixels.
command	Invoking function when user change content
cursor	Cursor of the mouse will change according to the given cursor name (arrow, dot etc).
font	Type of the font
exportselection	By default, if you select text within an Entry widget, it is automatically exported to the
clipboard	To avoid this exportation, use exportselection=0.
Fg	The color used to render the text.
highlightcolor	The color of the focus highlight when the checkbutton has the focus.
justify	If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT.
relief	With the default value, relief=FLAT, the checkbutton does not stand out from its background. You may set this option to any of the other styles .
Selectbackground	Background color of the selected text
selectborderwidth	The width of the border to use around selected text. The default is one pixel.
selectforeground	The foreground (text) color of selected text.
show	Normally, the characters that the user types appear in the entry. To make a .password. entry that echoes each character as an asterisk, set show="*".

state	The default is state=NORMAL, but you can use state=DISABLED to gray out the control and make it unresponsive. If the cursor is currently over the checkbutton, the state is ACTIVE.
Text variable	In order to be able to retrieve the current text from your entry widget, you must set this option to an instance of the StringVar class.
Width	The default width of a checkbutton is determined by the size of the displayed image or text. You can set this option to a number of characters and the checkbutton will always have room for that many characters.
Xscroll command	If you expect that users will often enter more text than the onscreen size of the widget, you can link your entry widget to a scrollbar.

**Example 14.3.** This program describes how to use a Tkinter entry in a program.

```

1 from Tkinter import *
2 tp=Tk()
3 la1=Label(tp,text="User ID")
4 la1.pack(side=LEFT)
5 en1=Entry(tp,bd=6)
6 en1.pack(side=RIGHT)
7 tp.mainloop()

```

When the above code is executed, it produces the output as show in the Figure 14.4:

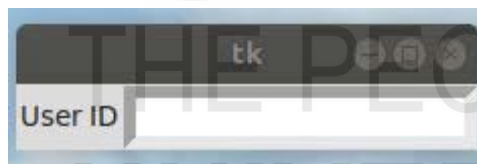


Figure 14.4 Text entry

### Using Label widget in window

This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.

### Syntax

w = Label ( master, option, ... )

### Parameters

**master:** This represents the parent window.

**options:** Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Option	Description
anchor	This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text in the available space.
Bg	The normal background color displayed behind the label and indicator.
Bitmap	Set this option equal to a bitmap or image object and the label will display that graphic.
Bd	The size of the border around the indicator. Default is 2 pixels.
Cursor	If you set this option to a cursor name (arrow, dot etc.), the mouse cursor will change to that pattern when it is over the checkbutton.
Font	If you are displaying text in this label (with the text or text variable option, the font option specifies in what font that text will be displayed.
Fg	If you are displaying text or a bitmap in this label, this option specifies the color of the text. If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap.
Height	The vertical dimension of the new frame.
Image	To display a static image in the label widget, set this option to an image object.
Justify	Specifies how multiple lines of text will be aligned with respect to each other: LEFT for flush left, CENTER for centered (the default), or RIGHT for right-justified.
Padx	Extra space added to the left and right of the text within the widget. Default is 1. pady Extra space added above and below the text within the widget. Default is 1.
Relief	Specifies the appearance of a decorative border around the label. The default is FLAT; for other values. text To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("\n") will force a line break.
Textvariable	To slave the text displayed in a label widget to a control variable of class StringVar, set this option to that variable.
Under Line	You can display an underline ( ) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining.

**Example 14.4. This program illustrates how to program using label and grid.**

1	import Tkinter as tk
2	m=tk.Tk()
3	tk.Label(m,text='First',bg="red").grid(row=0,column=0)
4	tk.Label(m,text='Second',bg="blue").grid(row=0,column=1)
5	tk.Label(m,text='Third',bg="green").grid(row=1,column=0)
	m.mainloop()

The output of the above program is illustrated in the figure 14.5. Remember you should write line 5 in one line.

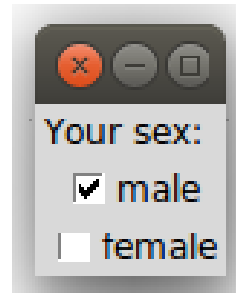


Figure 14.5 output of the above program

Setting rows and columns of grid are shown in following table. Any widget like entry, image can be arranged in grids.

Row=0, coloumn=0	Row=0, coloumn=1
Row=1, coloumn=0	Row=1, coloumn=1

### Using Check button widget : Checkbutton for selection

Output of running the code in example 14.5 is shown in the Figure 14.6



Figure 14.6 Check button entry

**Example 14.5 Program to describes how to program using label and checkbutton in Tkinter GUI.**

1	from Tkinter import * #import Tkinter module
2	m=Tk() # Tk window assigned to m
3	Label(m,text="Your sex: ").grid(row=0)
4	Checkbutton(m,text="male").grid(row=1)
5	Checkbutton(m,text="female").grid(row=2)
6	mainloop() #starts main loop- waiting for mouse and #key board

Line number 3 of the above code is a Label and “Your sex:” is displayed. There are two checkbuttons one for “male” and other for “female”. Mainloop() waits for mouse and keyboard.

Grid property set label and check buttons 3 top rows of the “m” window.

### Checkbutton for selection with align to the west

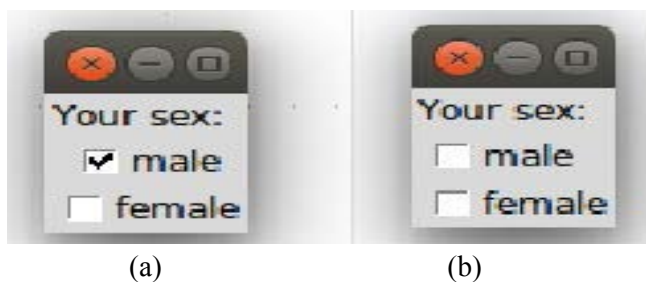
Checkbutton can be aligned using stick parameter in grid object. The program to explain the grid sticky option of checkbutton is given in example 14.6 and the output screen is shown in figure 14.7.

Example 14.6 This program explains how to program using checkbutton with grid sticky option.

```

1 from Tkinter import * #import Tkinter module
2 m=Tk() # Tk window assigned to m
3 Label(m,text="Your sex: ").grid(row=0,sticky=W)
4 Checkbutton(m,text="male",variable=var1).grid(row=1,sticky=W)
5 Checkbutton(m,text="female",variable=var2).grid(row=2,sticky=W)
6 mainloop() #starts main loop- waiting for mouse and key board

```



**Figure 14.7 Left alignments of check boxes**

Male and female checkbuttons of figure 14.7 (a) are not aligned and figure 11.7 (b) shows the aligned checkbuttons. This can be done with sticky option of grid manager. The above example has sticky=W i.e. aligned to left side. Likewise, following options also can be used.

N, E, S, W, NE, NW, SE, and SW

### Assigning checkbutton selection to variable

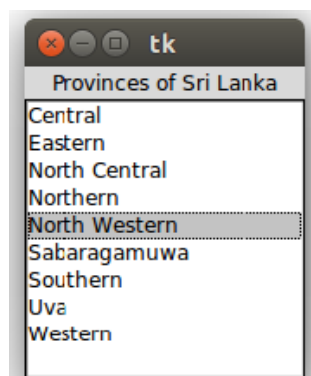
Example 14.7 This program describes how selections of checkbuttons are assigned to variables.

```

1 from Tkinter import * #import Tkinter module
2 m=Tk() # Tk window assigned to m
3 def var_states():
4     print var1.get()
5     print var2.get()
6 Label(m,text="Your sex: ").grid(row=0,sticky=W)
7 var1=IntVar()
8 Checkbutton(m,text="male",variable=var1).grid(row=1,sticky=W)
9 var2=IntVar()
10 Checkbutton(m,text="female",variable=var2).grid(row=2,sticky=W)
11 Button(m, text='Show', command=var_states).grid(row=4, sticky=W,
12     pady=4)
13 mainloop() #starts main loop- waiting for mouse and key board

```

Command of the button is binded to the var\_states function. Variable values in three different marks of checkbuttons are displayed in figure 14.8.



**Figure 14.8: Three different values in checkbutton**

## Listbox

Listbox is the Tkinter standard list box to facilitate a list. List of provinces in Sri Lanka created by using tkinter GUI library is shown in the figure 14.9. User can select a province from the list of provinces.

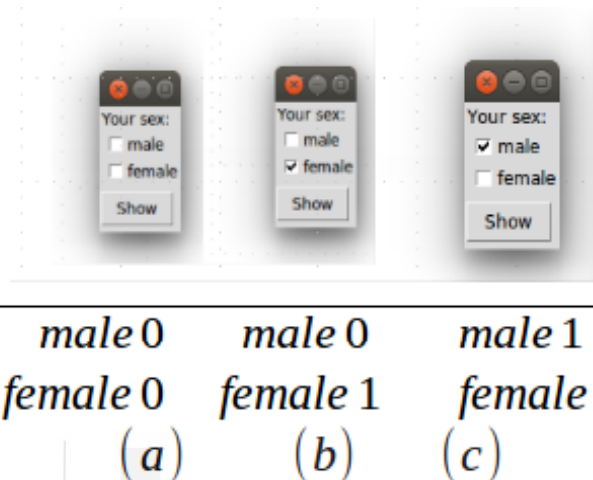


Figure 14.9: List of provinces, user can select either province from the list

Python code for the output of the Figure 14.9 is given in example 14.8.

**Example 14.8 program to describe listbox.**

```

1  from Tkinter import *
2  master = Tk()
3  l1=Label(master,text="Provinces of Sri Lanka")
4  l1.pack()
5  listbox = Listbox(master)
6  listbox.pack()
7  listbox.insert(END, "Central")
8  for item in ["Eastern",
9              "North Central",
10             "Northern",
11             "North Western",
12             "Sabaragamuwa",
13             "Southern",
14             "Uva",
15             "Western"]:
16      listbox.insert(END, item)
17  mainloop()

```

Line 3: l1 instance is derived from the Label class. Text of the label “l1” is “Provinces of Sri Lanka”.

```
l1=Label(master,text="Provinces of Sri Lanka")
```

Line 5: listbox instance is derived from the Listbox class.

```
listbox = Listbox(master)
```

Line 7: Append an item to the list.

```
listbox.insert(END, "Central")
```

Line 8 to 16: Append all other items to list.

```
for item in ["Eastern",
            "North Central",
            "Northern",
            "North Western",
            "Sabaragamuwa",
            "Southern",
            "Uva",
            "Western"]:
    listbox.insert(END, item)
```

## 14.7 GUI WITH WXPYTHON

WxPython is a free and open source cross platform GUI toolkit package. It can be downloaded from the official website <http://wxpython.org>. It consists of wxObject class, which is the base for all classes in the API. Control module contains all the widgets used in GUI application development. For example, wx.Button, wx.StaticText (analogous to a label), wx.TextCtrl (editable text control), etc.

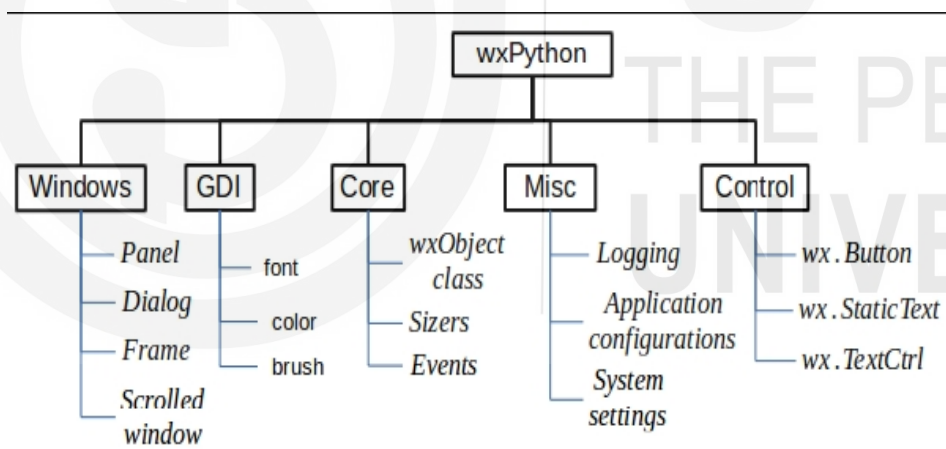


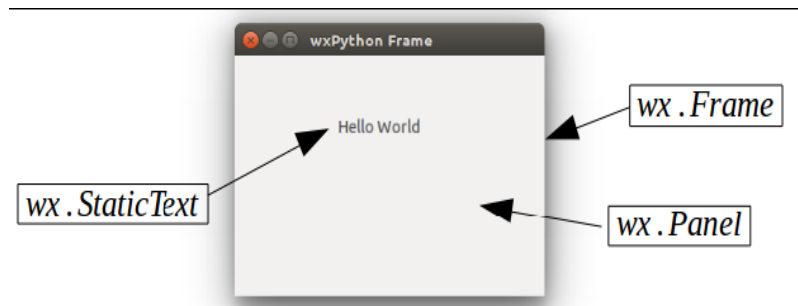
Figure 14.10 wxPython basic and sub modules

“wxPython” contains five basic modules, windows, Graphics Device Interface (GDI), Misc, Core and Controls. Basic modules and some of sub modules are depicted in figure 14.10.

### Basic wxPython GUI development

GUI in Figure 14.11 has a frame and a “Hello world” text.





**Figure 14.11. basic wxPython program output**

Output in Figure 14.11 is produced by the code in example 14.9.

Example 14.9 Program to explain to create wxPython GUI window

```
1 import wx
2 app = wx.App()
3 window = wx.Frame(None,title="wxPython Frame",size=(300,200))
4 panel=wx.Panel(window)
5 label=wx.StaticText(panel,label="Hello World",pos=(100,50))
6 window.Show(True)
7 app.MainLoop()
```

Let us analyse the above code to understand what it does.

Line 1: Import the wx module.

```
import wx
```

Line 2 : Define an object of Application class.

```
app = wx.App()
```

Line 3: Create a top level window as object of wx.Frame class. Caption and size parameters are given in constructor.

```
window = wx.Frame(None,title="wxPython Frame",size=(300,200))
```

Line 4: Although other controls can be added in Frame object, their layout cannot be managed. Hence, put a Panel object into the Frame.

```
panel=wx.Panel(window)
```

Line 5: Add a StaticText object to display 'Hello World' at a desired position inside the window.

```
label=wx.StaticText(panel,label="Hello World",pos=(100,50))
```

Line 6: Activate the frame window by show() method.

```
window.Show(True)
```

Line 7: Enter the main event loop of Application object.

```
app.MainLoop()
```

Line 3,4,5 of the above program include wx classes called wx.Frame, wx.Panel and wx.StaticText.

### Top level window classes (wx.Frame and wx.Panel) : wx.Frame

wx.Frame class is a top level window class. Higher level to this class is wx.Window. You can change its size and position. It is a container widget. It means that it can contain any window that is not a frame or dialog. wx.Frame has a title bar, borders and a central container area. The title bar and borders are optional. They can be removed by various flags.

wx.Frame Class has a default constructor with no arguments. It also has an overloaded constructor with the following parameters:

wx.Frame (parent, id, title, pos, size, style, name)

<code>window=wx.Frame(None, -1, "Hello", pos=(10,10), size=(300,200), style=</code>
<code>wxDEFAULT_FRAME_STYLE, name="frame")</code>

### wx.Panel

wx.Panel class is derived from the wx.Window class. This class is inserted to the wx.Frame class. Widgets such as button, text box etc. can be placed in this class.

### Sub classing the Frame -"Hello World" program with class

The GUI output in Figure 14.12 has a frame with the title "Hello World".

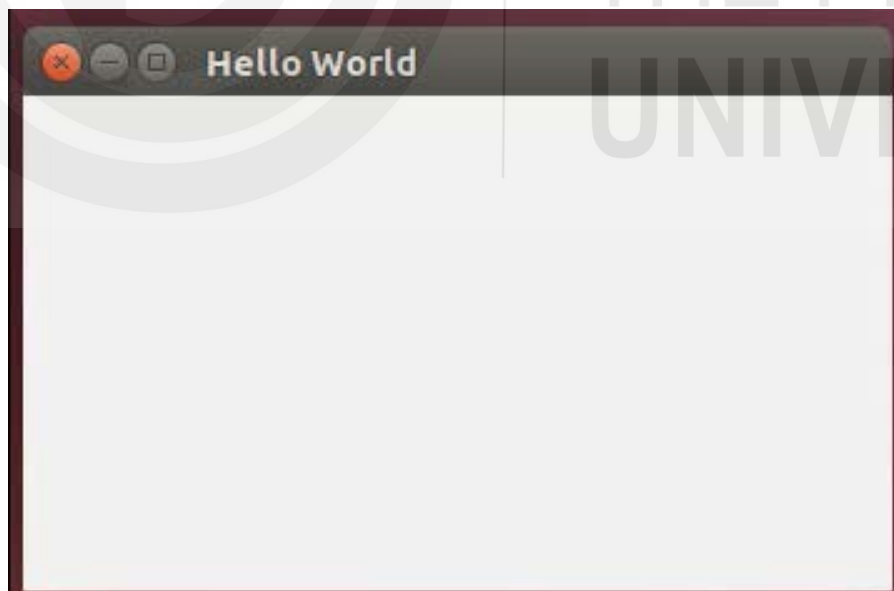


Figure 14.12: basic GUI of wxPython

Output of the Figure 14.12 is produced by running the program in example 14.10.

### Example 14.10 Program to create frame class called “Frame”.

```
1 import wx
2 class Frame(wx.Frame):
3     def __init__(self, title):
4         wx.Frame.__init__(self, None, title=title, size=(350,200))
5
6 app = wx.App(redirect=True)
7 top = Frame("Hello World")
8 top.Show()
9 app.MainLoop()
```

Let us analyse code lines of the example 14.10 to understand creation of frame class.

Line 2: class Frame is derived from the wx.Frame.

```
class Frame(wx.Frame):
```

Line 7: class Frame is called with the title “Hello World”

```
top = Frame("Hello World")
```

Line 3: The above title is received to the following class Frame and initiated with the title.

```
def __init__(self, title):
```

Line 4: title of the line 3 is received to the wx.Frame and it initiated with

```
wx.Frame.__init__(self, None, title=title, size=(350,200))
```

### Video 16: Python GUI with WxFrame

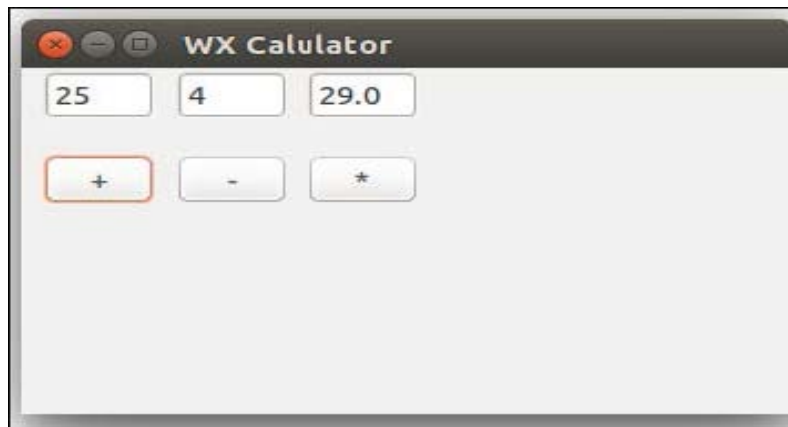
This video will demonstrate how to create a GUI with WxFramework. You may watch this Video to create the calculator example.

URL : <https://youtu.be/je7l6OMdjFU>



### Adding content to the frame with Calculator example

Calculator GUI in Figure 14.13 has 3 text boxes for inputting 2 numbers and for displaying result. There are three (3) buttons for calculating addition, subtraction and multiplication.



**Figure 14.13: Calculation of 2 variables**

Output of the Figure 14.13 is produced by running the code of the example 14.11.

**Example 14.11 program code describes how to use textcontrols and buttons of wxPython**

```

1  import wx
2  class wxcal(wx.Frame):
3      def __init__(self, title):
4          wx.Frame.__init__(self, None, title=title,
5 pos=(150,150), size=(350,200))
6          panel = wx.Panel(self)
7          self.fno=wx.TextCtrl(panel, value="", pos=(10, 2),
8 size=(50,-1))
9          self.sno=wx.TextCtrl(panel, value="", pos=(70, 2),
10 size=(50,-1))
11          self.result= wx.TextCtrl(panel, -1,"", pos=(130, 2),
12 size=(50,-1))
13          self.addbtn=wx.Button(panel, wx.ID_NONE, "+", pos =
14 (10,50), size=(50,-1))
15          self.addbtn.Bind(wx.EVT_BUTTON,self.addbtnclick)
16          self.subbtn=wx.Button(panel, wx.ID_NONE, "-", pos =
17 (70,50), size=(50,-1))
18          self.subbtn.Bind(wx.EVT_BUTTON,self.subbtnclick)
19          self.mulbtn=wx.Button(panel, wx.ID_NONE, "*", pos =
20 (130,50), size=(50,-1))
21          self.mulbtn.Bind(wx.EVT_BUTTON,self.mulbtnclick)
22          def addbtnclick(self,event):
23              self.result.SetValue(str(float(self.fno.GetValue())+
24 float(self.sno.GetValue())))
25          def subbtnclick(self,event):
26              self.result.SetValue(str(float(self.fno.GetValue())-
27 float(self.sno.GetValue())))
28          def mulbtnclick(self,event):
29              self.result.SetValue(str(float(self.fno.GetValue())*
30 float(self.sno.GetValue())))
31
32          app = wx.App(redirect=True)
33          top = wxcal("WX Calculator")
34          top.Show()
35          app.MainLoop()

```

Line 2: Class wxcal is created and its base class is wx.Frame.

```
class wxcal(wx.Frame):
```

Line 5: “panel” is a type of “wx.Panel” class. All the widgets such as text, buttons are placed in panel.

```
panel = wx.Panel(self)
```

Line 6 to 8: Two (2) text boxes (wx. TextCtrl) named “fno” and “sno” are placed in panel and their positions and sizes are specified as follows. There is another text box named “result” is also placed in specified position and size.

```
self.fno=wx.TextCtrl(panel, value="", pos=(10, 2), size=(50,-1))  
self.sno=wx.TextCtrl(panel, value="", pos=(70, 2), size=(50,-1))  
self.result= wx.TextCtrl(panel, -1,"", pos=(130, 2), size=(50,-1))
```

Line 10 to 11: Button for addition named “addbtn” is a wx.Button class type. Text of the button is “+” and position and size is as follows. This button has no identification ( wx.ID\_NONE). This button is bind to the method “addbtnclick” on the click event of the button (wx.EVT\_BUTTON ).

```
self.addbtn=wx.Button(panel, wx.ID_NONE, "+",pos=(10,50),size=(50,-1))  
self.addbtn.Bind(wx.EVT_BUTTON,self.addbtnclick)
```

Line 13 to 17: Program codes for buttons for subtraction and multiplication and their bindings.

```
self.subbtn=wx.Button(panel, wx.ID_NONE, "-",pos=(70,50),size=(50,-1))  
self.subbtn.Bind(wx.EVT_BUTTON,self.subbtnclick)  
  
self.mulbtn=wx.Button(panel, wx.ID_NONE, "*",pos=(130,50),size=(50,-1))  
self.mulbtn.Bind(wx.EVT_BUTTON,self.mulbtnclick)
```

Line 19 to 26: Methods to add, subtract and multiply values of fno and sno and assigned to result are coded here.

```
def addbtnclick(self,event):  
    self.result.SetValue(str(float(self.fno.GetValue()+float(self.sno.GetV  
alue()))))  
def subbtnclick(self,event):  
    self.result.SetValue(str(float(self.fno.GetValue()-  
float(self.sno.GetValue()))))  
def mulbtnclick(self,event):  
    self.result.SetValue(str(float(self.fno.GetValue()*float(self.sno.Get  
Value()))))
```

### WxPython Graphics Drawing Interface (GDI)

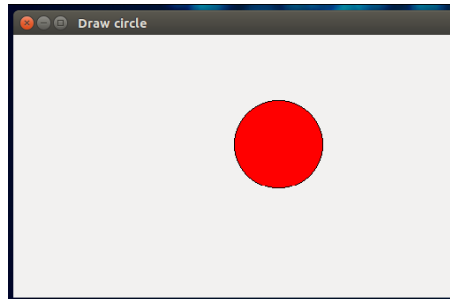
GDI in wxPython offers objects required for drawing shape, text and image like Colour, Pen, Brush and Font. It is used to interact with graphic devices such as monitor, printer or a file. It consists of 2D vector graphics, fonts and images. Objects called “device context” (DC) should be created to start drawing graphics. It represents number of devices in a generic way.

wx.DC classes are

```
# wxBufferedDC  
# wxBufferedDC  
# wxBufferedPaintDC  
# wxPostScriptDC  
# wxMemoryDC  
# wxPrinterDC  
# wxScreenDC
```

```
# wxClientDC
# wxPaintDC
# wxWindowDC
```

Drawing a circle with background colour of red in a window titled “Draw circle” is shown in Figure 14.14.



**Figure 14.14 Drawing circle using GDI**

Example 14.12 Program to display window and circle with red background	
1	import wx
2	class drawgeo(wx.Frame):
3	def __init__(self, parent, title):
4	super(drawgeo, self).__init__(parent, title=title, size=(500,300))
	self.InitUI()
5	
	def InitUI(self):
6	self.Bind(wx.EVT_PAINT, self.drawcle)
7	
	def drawcle(self, w):
8	dc = wx.PaintDC(self)
9	color=wx.Colour(255,0,0)
10	b=wx.Brush(color)
11	dc.SetBrush(b)
12	dc.DrawCircle(300,125,50)
13	app1 = wx.App()
14	top=drawgeo(None,"Draw circle")
15	top.Show()
16	app1.MainLoop()
17	

The above window and circle with red background can be programmed by using the following program. In this program, initially, create instance of wx app and activate a wx.frame custom class to draw the circle.

Now we will study program code lines in the above program. For easy reference, program is numbered at the left hand side column to the program.

1	import wx
	Import wxPython modules into the program
16	app1 = wx.App()
	Create an instance of wx app called app1
17	top=drawgeo(None,"Draw circle")

	Create an instance of the custom class of wx.Frame named “drawgeo” with the title “Draw circle”.
18	top.Show()
	Show the class “top”.
2	class drawgeo(wx.Frame):
	Create a wx.Frame class named “drawgeo”
3	def __init__(self, parent, title):
	Define the init method whose parameters are title and parameters belongs to super class “wx.Frame”.
4	super(drawgeo, self).__init__(parent, title=title,size=(500,300))
	Title of the super class is "Draw circle" and size is (500,300)
5	self.InitUI()
	Start InitUI method.
7	def InitUI(self):
	Define InitUI() method
8	self.Bind(wx.EVT_PAINT, self.drawcle)
	“drawcle” method of the class drawgeo is bound to the wx.EVT_PAINT event.
10	def drawcle(self,w):
	Define the method “drawcle()”
11	dc = wx.PaintDC(self)
	wx.PaintDC(self) is used to draw graphics in a client area.
12	color=wx.Colour(255,0,0)
	Create an instance of wx.Colour called “color” whose colour is (255,0,0). i.e. red.
1 3	b=wx.Brush(color)
	Create an instance of wx.Brush called “b” whose colour is “color”. wx.Brush is used to fill background colour of the object.
14	dc.SetBrush(b)
	Set the brush of the wx.PaintDC class called “dc”.
15	dc.DrawCircle(300,125,50)
	Draw the circle with x axis point, y axis point and radius

## wxPython GUI visual Designer tools

Coding GUI windows is a very difficult and time consuming effort. Therefore, there are few visual GUI designer tools available for rapid and quick GUI application development. They are wxFormBuilder, wxDesigner, wxGlade, BoaConstructor, gui2py.

---

### Activity14.1

- 1) Write a GUI program using Tkinter libraries to input two boolean values (0 or 1) and select a boolean operation from set of boolean operations AND, OR, NAND, NOR, XOR. Result should be printed in a text box next to the boolean selection box.
  - 2) Write a GUI program using wxPython libraries to input 2 boolean values (0 or 1) and select a boolean operation from set of boolean operations AND, OR, NAND, NOR, XOR. Result should be printed in a text box next to the boolean selection box.
- 

### Check Your Progress

Q-1 Write Tkinter lines of codes to embed following on a form

- a) Label
- b) Check Button
- c) Text entry
- d) Button

Q-2 Classify the modules and sub-modules in wxPython.

---

## 14.8 SUMMARY

---

In this unit you learned to identify elements of GUI, different types of python related packages for GUI development, GUI implement using Tkinter standard library and wxPython. Then you learned how to invoke widgets of GUI with methods in user defined classes.

---

## 14.9 FURTHER READING

---

- 1) kivy.org – official site
  - 2) TutorialsPoint (I) Pvt. Ltd., PYTHON programming language, [https://www.tutorialspoint.com/python/python\\_tutorial.pdf](https://www.tutorialspoint.com/python/python_tutorial.pdf)
- 

## 14.10 ANSWER TO CHECK YOUR PROGRESS

---

Ans-1 Refer section 14.6

Ans-2 Refer section 14.7



---

## UNIT 15 GUI PROGRAMMING USING KIVY LIBRARIES

---

### Structure

- 15.1 Introduction
- 15.2 Objectives
- 15.3 Basic GUI programming (user password GUI)
- 15.4 Kivy Layouts and Widgets
- 15.5 Kv language
- 15.6 Developing a Calculator using python and kivy
- 15.7 Develop a Calculator using python and kv language
- 15.8 GUI with check boxes
- 15.9 Summary
- 15.10 Further Reading
- 15.11 Answer to check your progress

---

### 15.1 INTRODUCTION

---

Developing GUI using kivy is different from Tkinter and wxPython because Kivy has a different architecture. You will learn how Kivy GUI libraries can be used in python programming by studying the given programming codes. Output of each programming code is given before the code snippet. Write and run codes and check whether the outputs are correct.

Python with kivy cross platform is a very useful method of implementing GUI for mobile application. Four different types of GUI programming examples are given in this unit to study different areas when you want to run a python GUI with Kivy libraries using widgets, layouts, running with only python or combined with python and Kv language.

---

### 15.2 OVERVIEW

---

Upon completion of this unit you will be able to:

- List type of layouts and widgets in Kivy libraries.
- Describe how to program by arranging and inserting widgets into grid layout.
- Describe calling methods when interacting with widgets.
- Describe programming with combining python and Kv language

## 15.3 BASIC GUI PROGRAMMING (USER PASSWORD GUI)

A sample user password entry GUI is shown in Figure 15.1 and it includes two labels and two text entry boxes in a two column gridlayout. Only the GUI shown in figure 15.1 is displayed and no other functions are executed when running the programming code.

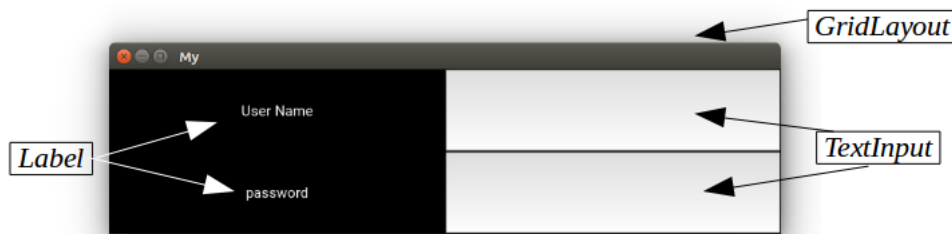


Figure 15.1: User password entry box using kivy

**Example 15.1** When you run the program given below, GUI of the figure 15.1 can be obtained.

```

1  from kivy.app import App
2  from kivy.uix.gridlayout import GridLayout
3  from kivy.uix.label import Label
4  from kivy.uix.textinput import TextInput
5  class LoginScreen(GridLayout):
6      def init__(self, **kwargs):
7          super(LoginScreen, self).__init__(**kwargs)
8          self.cols = 2
9          self.add_widget(Label(text='User Name'))
10         self.username = TextInput(multiline=False)
11         self.add_widget(self.username)
12         self.add_widget(Label(text='password'))
13         self.password = TextInput(password=True, multiline=False)
14         self.add_widget(self.password)
15  class MyApp(App):
16      def build(self):
17          return LoginScreen()
18  if __name__ == '__main__':
19      MyApp().run()

```

Now let us analyse the above program code. Line numbers are added for easy reference.

Line 1 to 4: importing the necessary Kivy GUI libraries (App, gridlayout, label and textinput).

```

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

```

Line 6: inbuilt class “GridLayout” has used for the base class for the class “LoginScreen”.

```

class LoginScreen(GridLayout):

```

Line 9: define the number of columns in the gridlayout. Here it is 2.

```
self.cols = 2
```

the 'User name' label, then, text input to input user name. These 2 widgets are inserted to first row of the two-column gridlayout. Then 'password' label and text input are inserted into 2<sup>nd</sup> row of the gridlayout.

```
self.add_widget(Label(text='User Name'))  
self.username = TextInput(multiline=False)  
self.add_widget(self.username)  
self.add_widget(Label(text='password'))  
self.password = TextInput(password=True,  
multiline=False)  
self.add_widget(self.password)
```

In kivy, GUI can be implemented with two types of reusable user interfaces, layouts and widgets.

---

## 15.4 KIVY LAYOUTS AND WIDGETS

---

### Layouts

There are 5 types of layouts in Kivy.

- 1) **GridLayout**: used to arrange widgets in a grid. Even if one dimension of the grid is given, Kivy can compute the size of the elements and arrange them.
- 2) **StackLayout**: used to arrange widgets adjacent to each other. It uses a set size in one dimension and does not try to make them fit within the entire space. This is useful to display widgets of the same size.
- 3) **AnchorLayout**: this layout considers only about children positions. It allows placing the children at a position relative to a border of the layout. Does not consider size\_hint.
- 4) **FloatLayout**: facilitate placing children with arbitrary locations and size, either absolute or relative to the layout size. Default size\_hint (1, 1) will make every child the same size as the whole layout, therefore this value should be changed if there are more than one child. To use absolute size, we can set size\_hint to (None, None). This widget considers pos\_hint, as a dict setting position relative to layout position.
- 5) **RelativeLayout**: Behaves similar to FloatLayout, except that children positions are relative to layout position, not to the screen.

### Widgets

Widgets are pieces of code (small programs) of user interface elements that provide different functions. Few examples of widgets are: file browser, buttons, sliders, and lists. Widgets may not be visible all the time and they receive MotionEvent.

## 15.5 KV LANGUAGE

Mobile GUI applications with kivy libraries can be coded in two different methods. In first method, functions, classes, widgets and their properties are coded in the same python file. This method is described with examples in section 15.1 and 15.4.

In the second method, functions and classes of application are coded in a python file. Extension of this file is .py. Widgets and their properties are coded in a kv file. Extension of this file is .kv. Therefore, logic of the application and its user interface can be separated clearly. GUI applications can be changed easily to suit user's requirements by doing this way. This method is described with examples in sections 15.5 and 15.6.

## 15.6 DEVELOPING A CALCULATOR USING PYTHON AND KIVY

In this calculator program, three text boxes are available for entering two numbers and other is used to print result. There are three buttons are available for addition (+), subtracting (-) and multiplication (\*). This calculator graphical user interface is shown in figure 2. When you press “+” button, 2 numbers will add result will be displayed in the third text box. Subtraction or multiplication will happen by pressing “-” and “\*” buttons.

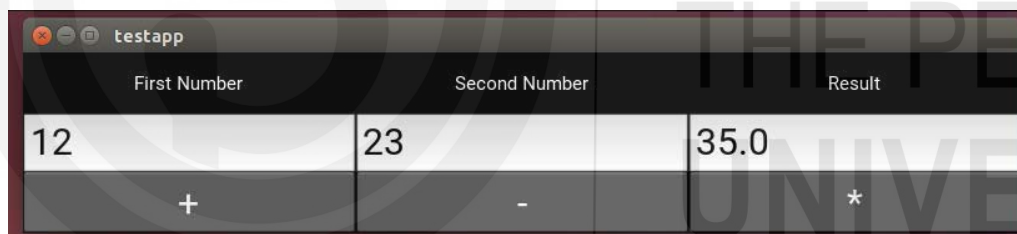


Figure 15.2: Calculation of 2 variables

Calculator in Figure 15.2 can be obtained by running the program code in example 15.2.

```
1 from kivy.app import App
2 from kivy.uix.button import Button
3 from kivy.uix.button import Label
4 from kivy.uix.gridlayout import GridLayout
5 from kivy.uix.textinput import TextInput
6 from kivy.uix.widget import Widget
7
8 class rootwd(GridLayout):
9     def __init__(self,**kwargs):
10         super().__init__(**kwargs)
11         self.cols = 3
12         self.add_widget(Label(text='First Number'))
13         self.add_widget(Label(text='Second Number'))
14         self.add_widget(Label(text='Result'))
15         #
16         self.fno = TextInput(multiline=False,font_size=30)
17         self.add_widget(self.fno)
18         #
19         self.sno = TextInput(multiline=False,font_size=30)
20         self.add_widget(self.sno)
21         #
22         self.tno = TextInput(multiline=False,font_size=30)
23         self.add_widget(self.tno)
24         self.display=self.tno
25         #
26         btnadd=Button(text="+",font_size=30)
27         self.add_widget(btnadd)
28         btnadd.bind(on_press=self.btnaddcal)
29
30         btnsub=Button(text="-",font_size=30)
31         self.add_widget(btnsub)
32         btnsub.bind(on_press=self.btnsubcal)
33
34         btnmul=Button(text="*",font_size=30)
35         self.add_widget(btnmul)
36         btnmul.bind(on_press=self.btnmulcal)
37
38     def btnaddcal(self,instance):
39
40         self.tno.text=str(float(self.fno.text)+float(self.sno.text))
41
42     def btnsubcal(self,instance):
43         self.tno.text=str(float(self.fno.text)-float(self.sno.text))
44
45     def btnmulcal(self,instance):
46
47         self.tno.text=str(float(self.fno.text)*float(self.sno.text))
48
49 class testapp(App):
50     def build(self):
51         return rootwd()
52 if __name__ == '__main__':
53     testapp().run()
```

### Example 15.2

Let's analyze the program given in example 15.2.

Line 8: Class rootwd belongs to the class GridLayout

```
class rootwd(GridLayout):
```

Line 11: Gridlayout rootwd has 3 columns.

```
self.cols = 3
```

Line 12 to 14: Add labels first number, second number and result to the first row of the rootwd grid layout.

```
self.add_widget(Label(text='First Number'))
self.add_widget(Label(text='Second Number'))
self.add_widget(Label(text='Result'))
```

Line 16 to 24: Add 3 text inputs named fno, sno and tno to the rootwd grid layout. Font size of each text input is set to 30 and multi line of text inputs are set to false. Line 17 is used to add text input to the rootwd grid layout.

```
self.fno = TextInput(multiline=False,font_size=30)
self.add_widget(self.fno)
#
self.sno = TextInput(multiline=False,font_size=30)
self.add_widget(self.sno)
#
self.tno = TextInput(multiline=False,font_size=30)
self.add_widget(self.tno)
self.display=self.tno
```

Line 26 to 28: Add button named btnadd to the grid layout and its text is “+” and font size is set to 30. Code in the line 28 bind the on\_press (when press on the button) to the method btnaddcal.

```
btnadd=Button(text="+",font_size=30)
self.add_widget(btnadd)
btnadd.bind(on_press=self.btnaddcal)
```

Line 38 to 39: **Method btnaddcal**, calculates the addition of two input texts “fno” and “sno” then result is assigned to the “tno”.

```
def btnaddcal(self,instance):
    self.tno.text=str(float(self.fno.text)+float(self.sno.text))
```

Lines 41 to 48 are same type of methods used to subtract and multiply two numbers.

## 15.7 DEVELOP A CALCULATOR USING PYTHON AND KV LANGUAGE

Output of the calculator example which is programmed using both Python and Kv language is shown in Figure 15.3.



Figure 15.3: Calculator using kv language

Two programming code files are used for this method. Name of the first file example 15.3(a) is **twonum.py** and name of the second file example 15.3(b) is **twonum.kv**. Now consider the Python program code in twonum.py

### Example 15.3 (a) twonum.py program code

```
1 import kivy
2 from kivy.app import App
3 from kivy.uix.gridlayout import GridLayout
4
5 class twonumgrid(GridLayout):
6     def cal(self,fnum,snum,op):
7         if op=="+":
8             self.display.text=str(float(fnum)+float(snum))
9         elif op=="-":
10            self.display.text=str(float(fnum)-float(snum))
11        elif op=="*":
12            self.display.text=str(float(fnum)*float(snum))
13
14 class twonumapp(App):
15
16     def build(self):
17         return twonumgrid()
18
19 if __name__ == '__main__':
20     twonumapp().run()
```

Line 5: Class “twonumgrid” is a “GridLayout” class.

```
class twonumgrid(GridLayout):
```

Line 6 to 12: method named “cal” to calculate addition, subtraction or multiplication according to the “op” input parameter. Input parameters to the method are fnum, snum and op (op is either + or – or \*). Variable values fnum, snum and op are received from the Kv file.

```
def cal(self,fnum,snum,op):
    if op=="+":
        self.display.text=str(float(fnum)+float(snum))
    elif op=="-":
        self.display.text=str(float(fnum)-float(snum))
    elif op=="*":
        self.display.text=str(float(fnum)*float(snum))
```

Now consider the Kv language file, twonum.kv file.

### Example 15.3 (b) twonum.kv program code

```

1  <CusButton@Button>:
2      font_size: 40
3
4  <CusText@TextInput>:
5      font_size: 35
6      multiline: False
7
8  <twonumgrid>:
9      id: twonumcal
10     display: result
11     rows: 2
12     padding: 10
13     spacing: 10
14
15     BoxLayout:
16         CusText:
17             id: fno
18         CusText:
19             id: sno
20         Label:
21             text: '='
22             font_size: 40
23         CusText:
24             id: result
25
26     BoxLayout:
27         CusButton:
28             text: "+"
29             on_press: twonumcal.cal(fno.text,sno.text,self.text)
30         CusButton:
31             text: "-"
32             on_press: twonumcal.cal(fno.text,sno.text,self.text)
33         CusButton:
34             text: "*"
35             on_press: twonumcal.cal(fno.text,sno.text,self.tex

```

Line 1 to 6: Button and TextInput widgets can be customized to suit our requirements and can be assigned to customize widgets. In this example, these two widgets are named as CusButton and CusText. This assignment is coded by using “@” sign. In this example, it is coded as CusButton@Button and CusText@TextInput. An instance of a root widget can be declared within <> and followed by:. For example <CusButton@Button>: and <twonumgrid>: in line 8.

```

<CusButton@Button>:
    font_size: 40

<CusText@TextInput>:
    font_size: 35
    multiline: False

```

Line 8 to 13: class “twonumgrid” is an instance of class GridLayout. In a widget tree, which displays information in a hierarchical structure like a tree, it is often needed to access or reference other widgets. The kv Language facilitates this access using ids. Id of the class twonumgrid is declared as “twonumcal”. Result of the two text inputs are displayed in the text input named “result”. Value of the result textinput is transferred to the class



twonumgrid by declaring to the display of the class. Number of rows in gridlayout is 2. Length of padding of widgets is declared as 10.

The **padding** argument tells Kivy how much space there should be between the Layout and its children, whereas the **spacing** arguments tell it how much spacing there should be between the children.

```
<twonumgrid>:  
    id: twonumcal  
    display: result  
    rows: 2  
    padding: 10  
    spacing: 10
```

Line 15 to 24: Boxlayout for the first row of the gridlayout is declared in these lines. Under the BoxLayout, there are two CusText text inputs which are identified by fno and sno. Another label is included for “=” and another CusText text input is included for displaying result and identified by “result”.

```
BoxLayout:  
    CusText:  
        id: fno  
    CusText:  
        id: sno  
    Label:  
        text: '='  
        font_size: 40  
    CusText:  
        id: result
```

Line 26 to 35: Second row of the gridlayout has another BoxLayout and it has three (3) CusButtons. Text of each CusButton is +, - and \*. Method “cal” of the class “twonumgrid” with its paramters in the twonum.py can

be invoked by the on\_press event. Input parameters are fno.text, sno.text and text of the Relevant CusButton.

```
BoxLayout:  
    CusButton:  
        text: "+"  
        on_press: twonumcal.cal(fno.text,sno.text,self.text)  
    CusButton:  
        text: "-"  
        on_press: twonumcal.cal(fno.text,sno.text,self.text)  
    CusButton:  
        text: "*"   
        on_press: twonumcal.cal(fno.text,sno.text,self.text)
```

---

## 15.8 GUI WITH CHECK BOXES

---

Various widgets are included in the Kivy library. Label, TextInput, CheckBox, Button are some common type of widgets.

GUI in figure 15.4 has 2 text entries for entering user's name and age. A group of check boxes is available for selecting title of the user (whether Mr.

or Mrs. or Ms). Information in the last line will be printed after typing above information and clicking on the Click button.

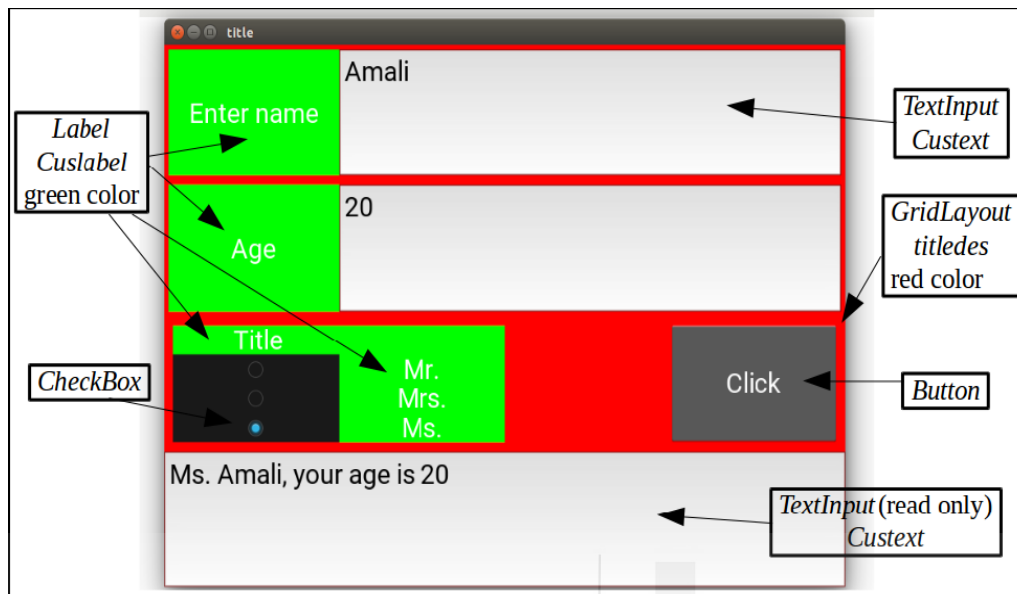


Figure 15.4: Text entry with check boxes

Now, let us analyse the two program codes title.py and title.kv. First, we will analyse the codes of title.py

#### Example 15.4 (a) title.py

```

1  import kivy
2  from kivy.app import App
3  from kivy.uix.gridlayout import GridLayout
4  class titledes(GridLayout):
5      def __init__(self, **kwargs):
6          super().__init__(**kwargs)
7          self.tt=""
8      def disptitle(self,name,age):
9          self.display.text=self.tt+" "+name+", your age is "+age
10         def cngtt(self,x):
11             self.tt=x
12     class titleApp(App):
13         def build(self):
14             return titledes()
15     titlapp=titleApp()
16     titlapp.run()

```

Line 1 to 3: import necessary kivy libraries

```

import kivy
from kivy.app import App
from kivy.uix.gridlayout import GridLayout

```

Line 4: class “titledes” is derived from the class “GridLayout” in the kivy library.

```

class titledes(GridLayout):

```

Line 5 to 6: Initialize the class “titledes”.

```

def __init__(self, **kwargs):
    super().__init__(**kwargs)

```

Line 7: Introduce a variable called “tt” in the class “titledes”. This variable is used to receive title of the user which is selected through the checkboxes.

```
self.tt=""
```

Line 8 to 9: Method to receive name and age form the title.kv file and they combined with the title (self.tt). Output is self.display.text. The display is declared in the .kv file under the class “titledes” gridlayout for the “strtxt” Textinput widget which is the output display place.

```
def disptitle(self,name,age):  
    self.display.text=self.tt+" "+name+", your age is "+age
```

Line 10 to 11: Method to receive label text relevant to activated checkbox and assign to the variable “tt” in the “titledes” GridLayout class.

```
def cngtt(self,x):  
    self.tt=x
```

**Example 15.4 (b) title.kv file**

```

1  <Cuslabel@Label>:
2      font_size: 30
3      canvas.before:
4          Color:
5              rgb: 0, 1, 0
6          Rectangle:
7              pos: self.pos
8              size: self.size
9  <Custext@TextInput>:
10     font_size: 30
11     multiline: False
12  <titledes>:
13     id: titledisp
14     display: strtxt
15     canvas.before:
16         Color:
17             rgb: 1, 0, 0
18         Rectangle:
19             pos: self.pos
20             size: self.size
21     rows: 4
22     BoxLayout:
23         padding: 5
24         Cuslabel:
25             text: "Enter name"
26             size_hint_x: None
27             width: 200
28         Custext:
29             id: nametxt
30     BoxLayout:
31         padding: 5
32         Cuslabel:
33             text: "Age"
34             size_hint_x: None
35             width: 200
36         Custext:
37             id: agetxt
38     GridLayout:
39         padding: 10
40         cols: 4
41         BoxLayout:
42             width: 2
43             canvas.before:
44                 Color:
45                     rgb: .1, .1, 0.1
46                 Rectangle:
47                     pos: self.pos
48                     size: self.size
49             orientation: 'vertical'
50             Cuslabel:
51                 text: "Title"
52                 size_hint_x: None
53                 width: 200
54             CheckBox:
55                 id: c1
56                 group: "title"
57                 on_active:
58 titledisp.cngtt(ttmr.text)
59             CheckBox:
60                 id: c2
61                 group: "title"
62                 on_active:
63 titledisp.cngtt(ttmrs.text)
64             CheckBox:
65                 id: c3
66                 group: "title"
67                 on_active:
68 titledisp.cngtt(ttms.text)
69             BoxLayout:
70                 width: 20

```

```

71 orientation: 'vertical'
72 Cuslabel:
73 Cuslabel:
74 id: ttmr
75 text: "Mr."
76 Cuslabel:
77 id: ttms
78 text: "Mrs."
79 Cuslabel:
80 id: ttms
81 text: "Ms."
82 BoxLayout:
83 BoxLayout:
84 Button:
85 font_size: 30
86 text: 'Click'
87 on_press:
88 titledisp.disptitle(nametxt.text,agetxt.text)
BoxLayout:
Custext:
id: strtxt
readonly: True

```

Second, lets analys the codes of the title.kv file.

Line 1 to 2: Declare the class called the “Cuslabel” which is an instance of the class “Label”. Its font size is 30.

```

<Cuslabel@Label>:
font_size: 30

```

Line 3 to 8: Declare Canvas, which is the root object used for drawing by a widget. These lines are focused to change color of the relevent widget. Here, it is “Cuslabel” Its color is green. The “rgb: 0,1,0” means red is 0, green is 1 and blue is 0.

```

canvas.before:
Color:
rgb: 0, 1, 0
Rectangle:
pos: self.pos
size: self.size

```

Line 9 to 11: “Custext” is an instance of “ TextInput” class. Its font size is set to 30 and multiline is off.

```

<Custext@TextInput>:
font_size: 30
multiline: False

```

Line 12 to 21: “titledes” is declared in the title.py which is a “GridLayout” class. This class in the title.kv file is identified by the “titledisp”. “display: strtxt” is the read only textinput to be displayed the output. Background color of the GridLayout is set to the red (rgb: 1,0,0). Number of rows of the GridLayout is 4.

```
<titledes>:
    id: titledisp
    display: strtxt
    canvas.before:
        Color:
            rgb: 1, 0, 0
        Rectangle:
            pos: self.pos
            size: self.size

    rows: 4
```

Line 22 to 29: BoxLayout is inserted to the first row of the GridLayout. It has a Cuslabel to display “Enter am” and a Custext to input the name.

```
BoxLayout:
    padding: 5
    Cuslabel:
        text: "Enter name"
        size_hint_x: None
        width: 200
    Custext:
        id: nametxt
```

Line 30 to 37: BoxLayout is inserted to the second row of the GridLayout. It has a Cuslabel to display “Age” and a Custext to input the age.

```
BoxLayout:
    padding: 5
    Cuslabel:
        text: "Age"
        size_hint_x: None
        width: 200
    Custext:
        id: agetxt
```

Line 38 to 40: Another 4 column GridLayout is inserted to third row to divide into 4 columns. Four columns are needed to insert checkboxes, labels of checkboxes, space and a button.

```
GridLayout:
    padding: 10
    cols: 4
```

Line 41 to 49: BoxLayout is inserted to the first column of the third row. Widgets can be inserted vertically because orientation is 'vertical'. Color of the BoxLayout also changes.

```
BoxLayout:
    width: 2
    canvas.before:
        Color:
            rgb: .1, .1, 0.1
        Rectangle:
            pos: self.pos
            size: self.size
    orientation: 'vertical'
```

Line 50 to 65: first row of the first column of the third row is allocated to label the “Title”. Other three rows are filled with three (3) CheckBoxes. Active CheckBox call the “cngtt” method of the ttle.py file with relevant text parameter of the Label of checkbox. All the CheckBoxes are grouped as “title”. When they are grouped, only one checkbox can be selected at a time.

```
Cuslabel:
    text: "Title"
    size_hint_x: None
    width: 200
CheckBox:
    id: c1
    group: "title"
    on_active: titledisp.cngtt(ttmr.text)
CheckBox:
    id: c2
    group: "title"
    on_active: titledisp.cngtt(ttmrs.text)
CheckBox:
    id: c3
    group: "title"
    on_active: titledisp.cngtt(ttms.text)
```

Line 66 to 78: BoxLayout is inserted to second column of the third row. Widgets to this BoxLayout can be inserted vertically because orientation is 'vertical'. Cuslabel widgets are inserted to display text of checkboxes.

```
BoxLayout:
    width: 20
    orientation: 'vertical'
Cuslabel:
Cuslabel:
    id: ttmr
    text: "Mr."
Cuslabel:
    id: ttmrs
    text: "Mrs."
Cuslabel:
    id: ttms
    text: "Ms."
```

Line 79 to 84: Empty BoxLayout is inserted to the third column of the third row. Then BoxLayout is inserted to fourth column of the third row for inserting the Button for “Click”. When the Button is pressed, method “disptitle” will call with 2 parameters “nametxt.text” and “agetxt.text”.

```
BoxLayout:
BoxLayout:
    Button:
        font_size: 30
        text: 'Click'
        on_press: titledisp.disptitle(nametxt.text,agetxt.text)
```

Line 85 to 88: BoxLayout is inserted to the fourth row of the GridLayout. This BoxLayout has readonly Custext to display output.

```
BoxLayout:
    Custext:
        id: strtxt
        readonly: True
```

### **Activity15.1**

Write a GUI program using kivy libraries and kv language to input 2 boolean values (0 or 1) and select a boolean operation from set of boolean operations AND, OR, NAND, NOR, XOR. Result should be printed in a text box next to the boolean selection box.

### **Check Your Progress**

- Q-1 Discuss various types of layouts used to develop GUI in Kivy.
- Q-2 What are widgets? How widgets differs from layouts?
- Q-3 Discuss various methods for coding the mobile GUI applications.

## **15.9 SUMMARY**

In this unit you learned how to create a graphical user interface with kivy libraries using only Python as well as using both Python and Kv language. We also discussed calling methods in Python on actions of Kivy widgets.

### **15.10 FURTHER READING**

- 1) Kivy Documentation Release 1.10.1.dev0,  
<https://media.readthedocs.org/pdf/kivy/latest/kivy.pdf>
- 2) kivy.org – official site

### **15.11 ANSWER TO CHECK YOUR PROGRESS**

- Ans-1 Refer section 15.4
- Ans-2 Refer section 15.4
- Ans-3 Refer section 15.5



---

## Appendix 1    Answers to Activities given in Python book

---

### Unit 01

#### *Activity1.1*

- 1) Python is a programming language which has a very simple and consistent syntax. It allows beginners to concentrate on important programming skills. Python helps to introduce basic concepts such as loops and procedures quickly to students.
- 2) There is an interactive interpreter in Python which helps students to test language features while they're programming. Students would be able to see both windows (the interpreter running and their program's source) at the same time.
- 3) Good IDEs are available for Python.
- 4) Python language is intuitive and fun. Since Python is an open source programming language, it reduced up-front project costs as well.

#### *Activity1.2*

Nowadays, Python is used in many application domains to cater for Web and Internet Development, Scientific and Numeric applications, Education applications, Desktop GUIs and Software Development.

#### *Activity1.3*

See HELP documents if you cannot get the PATH setup correctly. Issues vary depending on the directory you installed Python and the operating system.

### UNIT 02

#### *Activity2.1*

Typing each statement will give you following results.

```
>>> 55
55

>>> x=55
>>> x+1
56
```

In a new file from IDE type; 55

```
x = 55
print(x) x + 1 print(x)
```

Save the file and select 'run module' from Run. Output was;

```
===== RESTART: C:/Users/sarala/Documents/COL/Check Your
Progress2- printscript1.py =====
```

```
55
55
>>>
```

### **Activity 2.2**

```
>>> width = 42
>>> height = 14
>>> width/3 14.0
>>> width/3.0 14.0
>>> height/3 4.6666666666666665
>>> 13 + 25* 10
263
```

### **Activity 2.3**

```
1. >>>h
    =6
    >>>r =

2
>>> pi = 3.141592
>>> volume= pi*r*r*h
>>>print(volume)
```

## **UNIT 03**

### **Activity3.1** #Program to print even numbers between 20 and 60

```
n = 20
while n <= 60 :
    n = n + 2
    print(n)
print('All even numbers between 20 to 60 are printed')
```

### **Activity3.2**

```
# Use of For loop
import math
numberlist = [1,2,3,4,5]
for no in numberlist:
    print(no, "cube of ", no, " is", (math.pow(no,3)))
```

### **Activity3.3**

```
>>> car = ('Toyota', 'Aqua', 2015, 'TA-181')
>>> print(car)
('Toyota', 'Aqua', 2015, 'TA-181')
```

## **UNIT 04**

### **Activity4.1**

```
def findPrime(no):
    i = 0 divisorList=[2,3,5,7,11,13,17,19]

    for j in divisorList:
```

```
if (no != j) and (no % j == 0)
: i = 1
if i == 1:
print("False")
else:
print ("True") findPrime(4)
findPrime(1)
findPrime(19)
Required argument type is used
```

#### **Activity4.2**

```
no = 397.234567
print(round(no,3))
397.235
```

#### **Activity4.3**

```
>>> g = lambda x : x**2
>>> print (g(9))
81
```

### **UNIT 05**

#### **Activity5.1**

```
flower = "jasmine" index = len(flower)- 1
while index >= 0:
letter = flower[index] print (letter)
index = index - 1
```

#### **Activity5.2**

```
>>>
flow
er[:]

'jas
min
e'
```

#### **Activity5.3**

(i)

```
str1 = 'Divya asked, '
str2 = 'Great! Then can you let me have all the mangos, limes,
oranges and apples?'
str3 = str1 + str2 print(str3) index = 0
while index < len(str3):
```

```

letter = str3[index]
    if letter.isalpha():
        print (letter)
else:
    print(' ')
index = index + 1

```

(ii)

```

str1 = 'Divya asked, '
str2 = 'Great! Then can you let me have all the mangos, limes, oranges and apples?'
str3 = str1 + str2
print(str3)
index = 0
while index < len(str3):
    letter = str3[index]
    if letter.isalpha():
        print (letter, end = " ")
    else:
        print(' ', end = " ")
    index = index + 1

```

## UNIT 06

### Activity6.1:

Write a Student class which contains studentID, lastName, courseID.  
Input values to one object of type student and print the values.

**class Student:**

```

def __init__(self, studentID, lastName, courseID):
    self.studentID = studentID
    self.lastName = lastName
    self.courseID = courseID

```

```

>>> s1 = Student();
>>> print(s1.studentID, s1.lastName, s1.courseID)

```

### Activity6.2:

Write an init method for the Bikeclass that takes gear and speed  
and initialize them.  
# inside class Bike:

```

def __init__(self, gear=1, speed=0):
    self.gear = gear
    self.speed = speed

```

### Activity6.3:

Write a \_\_str\_\_ method for the Bike class and print it.

# inside class Bike:

```

def __str__(self):
    return('Gear %d%, speed %d%(self.gear, self.speed))
>>> myBike = Bike(2,35)
>>> print(myBike)

```

## UNIT 07

### Activity7.1:

- 1) Write a Person Class. Make another class called Student and inherits it from Person class.

- 2) Define few attributes that only have with Student class, such as school they are associated with, graduation year, GPA etc.
- 3) Create an object called student  
Set some attribute values for the student, that are only coded in the Person class and another set of attribute values for the student, that are only in the Student class.
- 4) Print the values for all of these attributes.

**Activity 7.2:**

A CEO buys a car. Later on the CEO buys two new cars BMW and a Mercedes. There is a driver for the CEO who chooses a car to drive to the office.

- 1) Identify the classes involved in this scenario.
- 2) Select appropriate superclass and subclasses
- 3) Implement move method inside the superclass.
- 4) Invoke the move method in superclass by creating instances of subclasses from a sub class.
- 5) Implement the move method inside the subclasses.
- 6) Override the move methods by creating instances of sub class.

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

class CEO(Person):
    def __init__(self, xxx, *args, **kwargs):
        self.xxx = xxx
        super(CEO, self).__init__(*args, **kwargs)

class driver(Person):
    PERMANENT, TEMPORARY = range(2)
    def __init__(self, employment_type, *args, **kwargs):
        self.branch = []
        self.employment_type = employment_type
        super(driver, self).__init__(*args, **kwargs)
    def enrol(self, yyy):
        self.branch.append(yyy)

class vehicle:
    def __init__(self, model, make,color, gear, *args, **kwargs):
        self.model = model
        self.make = make
        self.idno = color
        self.gear = gear

    def move(self, gear):
        print("gear given is :", gear)

class car(vehicle):
    def __init__(self, currFuel, miles, *args, **kwargs):
```

```

self.currFuel = currFuel
self.miles = miles
super(car, self).__init__(*args, **kwargs)
def move(self, miles):
    fuelNeeded = miles/10
    if self.currFuel <= fuelNeeded:
        print("Need re-fueling")
    else:
        print("Has sufficient fuel, can move")
        return {"changeSpeed", self.gear}
BMW=car(23,34, 'x','xx','brown', 2)
BMW.move(34)

```

## UNIT 08

### *Activity8.1:*

List the different types of errors and explain how you can identify them separately.

Syntax errors are produced by Python when it is translating the source code into byte code. They usually indicate that there is something wrong with the syntax of the program.

Runtime errors are produced by the interpreter if something goes wrong while the program is running. Most runtime error messages include information about where the error occurred and what functions were executing.

Semantic errors are problems with a program that runs without producing error messages but doesn't do the right thing. In other words the program does execute the correct logic.

### *Activity8.2:*

What are exceptions and why is it important to handle them appropriately. State with examples.

Errors detected during execution are called exceptions and are not unconditionally fatal.

Essentially, exceptions are events that modify program's flow, either intentionally or due to errors. They are special events that can occur due to an error, e.g. trying to open a file that doesn't exist, or when the program reaches a marker, such as the completion of a loop.

Example: When something goes wrong during the runtime, Python prints a message that includes the name of the exception, the line of the program where the problem occurred, and a traceback.

### *Activity8.3:*

Describe what user-defined exceptions are.

These are exceptions that Python allows the programmer to create based on his/her requirements. However it is better to check before creating an exception if there is already an existing one.

## UNIT 09

### *Activity9.1:*

Explain the difference between white box testing and black box testing.

White Box testing is where the internal structure of the system such as control structures of the program are tested and the Black Box testing is where the functionalities of the system is tested without going into the details of the implementation.

### *Activity9.2:*

Test suite: A test suite is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.

Explain the concepts used in unit testing.

unittest supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. A few of the concepts that is supported by Python includes:

Test fixture: A test fixture represents the preparation needed to perform one or more tests, and any associate cleanup actions.

Test case: A test case is the smallest unit of testing. It checks for a specific response to a particular set of inputs.

### *Activity9.3:*

Write a test case for a string method that test for a "FOOD".

Please refer to the unit testing video.

## UNIT 10

### *Activity10.1:*

What is a debugger framework (bdb) and state each functions it handles with examples.

The bdp module handles basic debugger functions, like setting breakpoints or managing execution via the debugger.

The following syntax is used to define the exception which would be raised by the bdb class for quitting the debugger.

```
exception bdb.BdbQuit
```

### *Activity10.2:*

What is a Python debugger (pdb) and what are the debugging functionalities it supports.

The module pdb defines an interactive source code debugger for Python programs. It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

### **Activity10.3:**

What is a profile and by referring to the examples above(in the text book), profile a function that takes in a single argument.

It provides a very brief overview, and allows a user to rapidly perform profiling on an existing application.

To profile a function that takes a single argument, you can do:

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

## **UNIT 11**

### **Activity11.1:**

*Have to write the database as instructed*

## **UNIT 12**

### **Activity12.1:**

You need to research on the internet to find out the most popular Python libraries or packages available as open source repositories to use in different domains such as data science, image processing, information retrieval, data manipulation etc. Few such libraries are given below.

SQLAlchemy: This is used as the Python SQL toolkit and Object Relational Mapper.

Pillow : This is a Python Imaging Library(PIL) which adds image processing capabilities to the Python interpreter.

Scrapy : A collaborative framework for extracting the data required from websites.

NumPy: This is the fundamental package for scientific computing with Python.

Matplotlib: This is a Python 2D plotting library which produces quality figures.

## **UNIT 13**

### **Activity13.1:**



The Kivy launcher can be used to run the Kivy applications on Android devices without compiling them. Kivy launcher runs your Kivy application when you copy it inside the SD Card of your device.

To install the Kivy launcher, you must go to the Kivy Launcher page on the Google Play Store. Then click on Install and select your phone. If not you can go to <https://kivy.org/#download> and install the APK manually.

Once the Kivy launcher is installed, you can put your Kivy applications in the Kivy directory in your external storage directory. Often the application is available at /sdcard even in devices where this memory is internal. For an example /sdcard/kivy/<your application>

<your application> should be a directory containing your main application file(e.g. main.py) and a text file (android.txt) with the contents given below.

```
title= <Application Title>
author=<Your name>
orientation=<portrait|landscape>
```

## UNIT 14

### Assignment 14.1: Answer

```
from Tkinter import *
root = Tk()
def evaluateand():
    if n0.get()=="1" and n1.get()=="1":
        out.config(text="1")
        out.update_idletasks()
    else:
        out.config(text="0")
        out.update_idletasks()
def evaluateor():
    if n0.get()=="1" or n1.get()=="1":
        out.config(text="1")
        out.update_idletasks()
    else:
        out.config(text="0")
        out.update_idletasks()
root.geometry("300x100")
n0=Entry(root,width=2)
n1=Entry(root,width=2)
result=Entry(root,width=2)
n0.delete(0)
n1.delete(0)
n0.place(x=20,y=1)
n1.place(x=20,y=40)
out=Label(root,text="")
out.place(x=150,y=20)
logicand= Button(root, text="AND",command=evaluateand)
logicand.place(x=50,y=20)
logicor= Button(root, text="OR",command=evaluateor)
logicor.place(x=50,y=50)
root.mainloop()
```



```

import wx
class logicex(wx.Frame):
    def __init__(self, title):
        wx.Frame.__init__(self, None, title = title, size = (400,200))
        panel = wx.Panel(self)
        self.n0=wx.TextCtrl(panel,value="",pos=(10,2))
        self.n1=wx.TextCtrl(panel,value="",pos=(10,100))
        self.result=wx.TextCtrl(panel,value="",pos=(300,50))
        lg = ['AND', 'OR', 'NAND', 'NOR', 'XOR']
        self.combo = wx.ComboBox(panel,choices = lg,pos=(90,50))
        self.combo.Bind(wx.EVT_COMBOBOX, self.oncombo)

    def oncombo(self,event):
        self.lsel = self.combo.GetValue()
    if self.lsel=="AND":
        if self.n0.GetValue()=="1" and self.n1.GetValue()=="1":
            self.result.SetValue("1")
        else:
            self.result.SetValue("0")

app = wx.App()
top=logicex("Logic Example")
top.Show()
app.MainLoop()

```



## UNIT 15

### Assignment 15.1: Answer andk.py file

#### andk.kv file

```

import kivy
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.properties import ObjectProperty
class andkgrid(GridLayout):
    def callog(self,n0,n1,lg):
        if lg=="AND":
            if n0=="1" and n1=="1":
                self.display.text=str(1)
            else:
                self.display.text=str(0)

class andkapp(App):
    def build(self):
        return andkgrid()

if __name__ == '__main__':
    andkapp().run()

```

```
*
<CusButton@Button>:
    font_size: 40
<CusText@TextInput>:
    font_size: 35
    multiline: False
<andkgrid>:
    id: andkcal
    display:result
    rows:3
    padding: 10
    spacing: 10
    BoxLayout:
        CusText:
            id: q0
        Label:
            text: "
            font_size: 40
        Label:
            text: "
            font_size: 40
        BoxLayout:
            spacing: 10
            Label:
                text: "
                font_size: 40
            CusButton:
                text: "AND"
                on_press: andkcal.callog
        (q0.text,q1.text,self.text)
        CusText:
            id: result
        BoxLayout:
            spacing: 10
```

```
CusText:
    id: q1
Label:
    text: "
    font_size: 40
Label:
    text: "
    font_size: 40
```

\*-Note that this line code belongs to the upper line