e

UNIT 12 ROLE OF PYTHON IN MOBILE APPLICATION DEVELOPMENT

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. Otherewise, 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 Windowsdevelopment, 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.

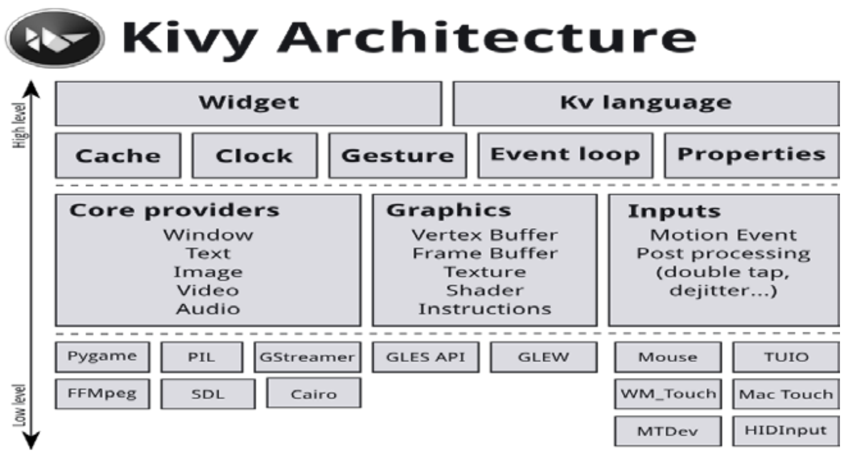

Figure 12.1: Kivy Architecture

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

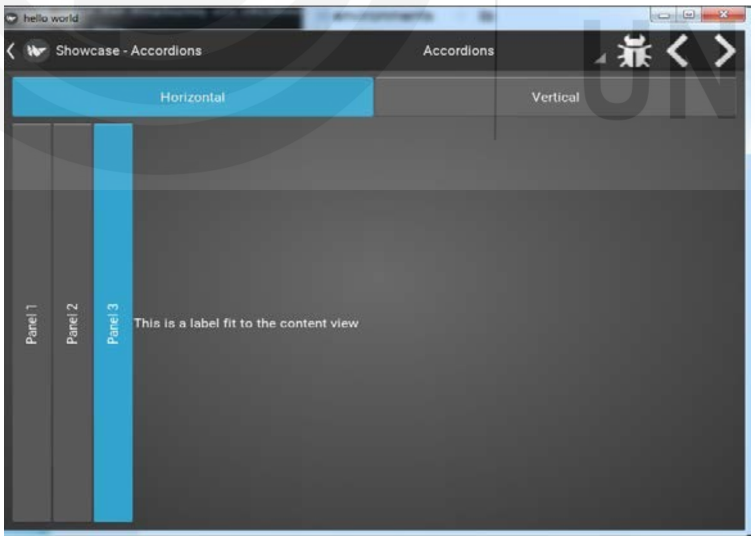

Figure 12.2 : Output of the main.py program

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.pyfor 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.

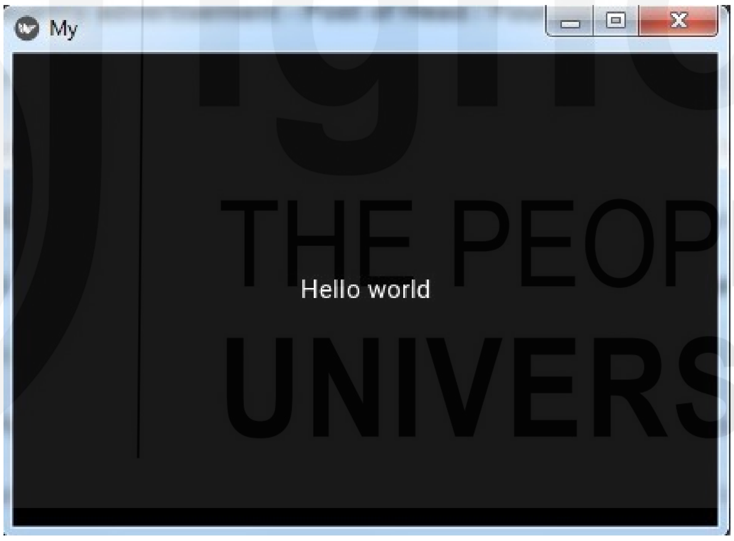

Figure 12.3: Output of first.py

- In order to use Kivy, it's required to importkivy 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 uixmodule 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 Labelwith 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.

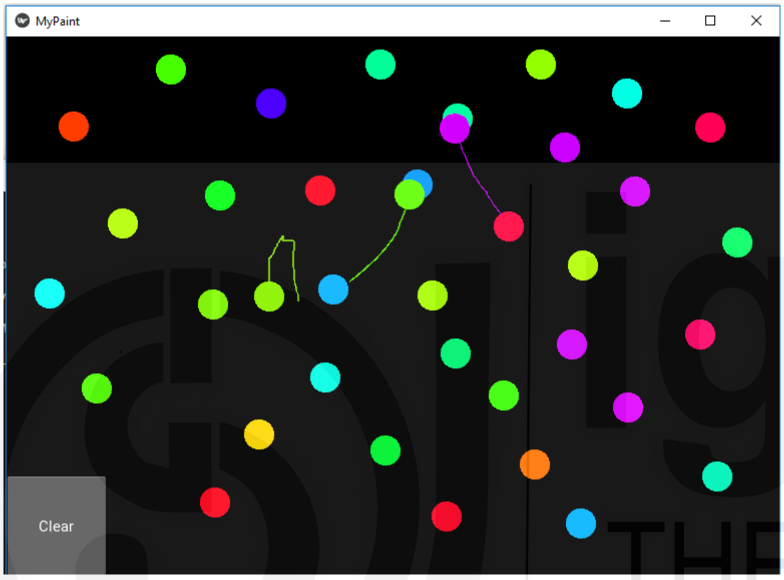

Figure 12.4: Output of **paint.py**

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 Buttonclass.
- 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_releaseevent (which is fired when the button is pressed and then released) to the callback

function clear_canvasdefined on below on lines 25 & 26.

We set up the widget hierarchy in line 22 and 23 by making both the painter and the clearbtnchildren 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.

Activity12.1

- 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 Waels (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