**Unit 1: INTRODUCTION TO Android Programming**

**1.1 What is Android?**

=> Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets.

**1.2 History and Versions:**

| Sr. No | Name | Version | Api Level |
| --- | --- | --- | --- |
| 1 | Android 1.0(Astro Boy) | 1.0 | 1 |
| 2 | Android 1.1(Bender) | 1.1 | 2 |
| 3 | Android Cupcake | 1.5 | 3 |
| 4 | Android Donut | 1.6 | 4 |
| 5 | Android Eclair | 2.0 | 5,6,7 |
| 6 | Android Froyo | 2.2 | 8 |
| 7 | Android Gingerbread | 2.3 | 9,10 |
| 8 | Android Honeycomb | 3.0 | 11,12,13 |
| 9 | Android Ice Cream Sandwich | 4.0 | 14,15 |
| 10 | Android Jelly Bean | 4.1 | 16,17,18 |
| 11 | Android Kitkat | 4.4 | 19,20 |
| 12 | Android Lollipop | 5.0 | 21,22 |
| 13 | Android Marshmallow | 6.0 | 23 |
| 14 | Android Nougat | 7.0 | 24,25 |
| 15 | Android Oreo | 8.0 | 26,27 |
| 16 | Android pie | 9 | 28 |
| 17 | Android 10 (Quince Tart) | 10 | 29 |
| 18 | Android 11 (Red Velvet Cake) | 11 | 30 |
| 19 | Android 12 (Snow Cone) | 12 | 31,32 |
| 20 | Android 13 (Tiramisu) | 13 | 33 |

**1.3 Android Architecture**

=>

Android architecture is a software stack of components to support mobile device needs. Android software stack contains a Linux Kernel, a collection of c/c++ libraries which are exposed through an application framework services, runtime, and application.

Following are main components of android architecture those are:

1. **Applications:**

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.

1. **Android Framework:**

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.

It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

1. **Android Runtime:**

The Android Runtime environment is one of the most important parts of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

1. **Platform Libraries:**

The Platform Libraries include various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide support for android development.

i) **Media** library provides support to play and record audio and video formats.

ii) **Surface manager** responsible for managing access to the display subsystem.

iii) **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.

iv) **SQLite** provides database support and FreeType provides font support.

v) **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.

vi)**SSL** (Secure Sockets Layer) is security technology to establish an encrypted link between a web server and a web browser.

1. **Linux Kernel:**

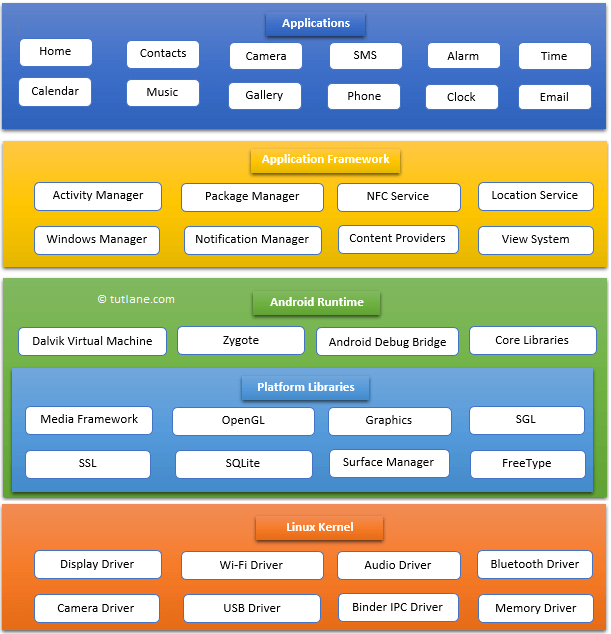
Linux Kernel is the heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.

The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

In these components, the Linux Kernel is the main component in android to provide its operating system functions to mobile and Dalvik Virtual Machine (DVM) which is responsible for running a mobile application.

Following is the pictorial representation of android architecture with different components.

* **Security**: The Linux kernel handles the security between the application and the system.
* **Memory Management**: It efficiently handles the memory management thereby providing the freedom to develop our apps.
* **Process Management**: It manages the process well, allocates resources to processes whenever they need them.
* **Network Stack**: It effectively handles the network communication.
* **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.



**1.4 Basic Building Blocks (Android Components)**

=>

The fundamental building blocks of Android are Activities, Services, Broadcast Receivers and Content Providers. Interchange in data between these blocks is handled by Intents which can be either Explicit or Implicit intents.

**Activities**

It Control the UI and handle the user interaction to the Android App screen,it represents a single screen with a user interface.Activity performs actions on the screen, For example, Medium application might have one activity that shows a list of articles while another activity to read the Notifications and another activity for viewing user profile.

**Services**

The Service is another building block of Android applications. They handle background running tasks associated with an application. It is a program that can run in the background for an infinite period and it does not provide a UI. A good example of service is the music player app, let us consider Google play Music which keeps on playing music while the main android app activity is destroyed or closed.

**Broadcast Receivers**

They handle communication between core Android OS and applications running on the surface layer. The application modules communicate with each other using the broadcast receiver. Messages can be initiated by the system or the app. For example, if a user is using your app and a phone call comes in, the Broadcast Receiver interface will notify your app.

**Content Providers**

They handle data interchange and database related components. This provides a uniform singular interface to the content & data. The content can be from your app being exposed to the system or other apps or it could be system content for example, pictures on your phone, contact lists, or files in the file system. It provides a consistent interface to retrieve/store data via external data sources or databases.

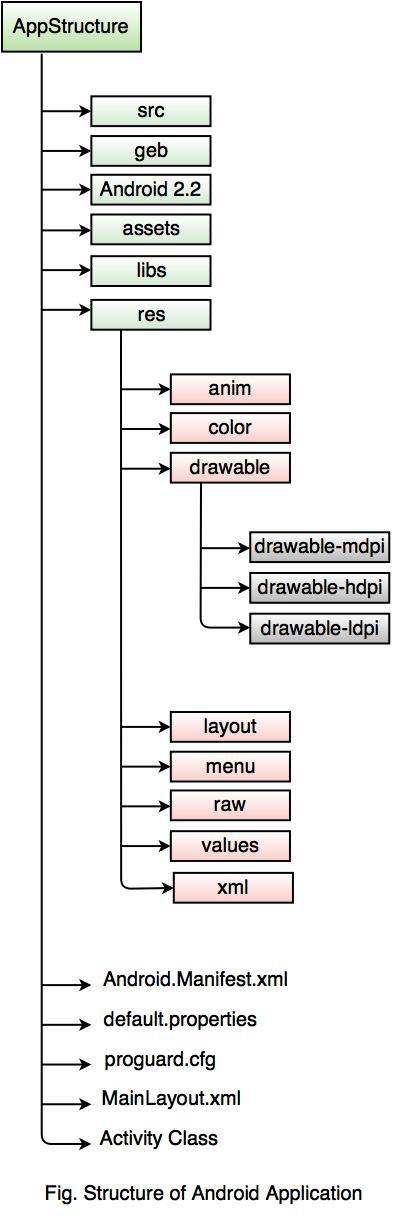
**1.5 Android API Levels**

=>

Refer: <https://apilevels.com/>

**1.6 Application Structure**

**=>**

****

**1.7 First Hello World Program**

=>

Refer program available on GitHub

**Unit 2: ACTIVITY, INTENT AND LAYOUT**

**2.1 Introduction to Activity**

=>

An activity is the single screen in android.

**2.2 Activity Life Cycle**

=>

****

1. onCreate()
2. onStart()
3. onResume()
4. onPause()
5. onStop()
6. onRestart()
7. onDestroy()

Example: Refer github for activity life cycle demo

**2.3 Introduction to Intent**

=> An Intent is a messaging object you can use to request an action from another app component.

Let’s look upon the informal way of defining Intents. You can think of Intents as a messaging service that is used to communicate between various components of the Android application. For example, if you want to send some message from Delhi to Mumbai using the Post Office facility then you can do so by buying an Envelope and then pass the message in the Envelope and send the message to the desired location.

**2.4 Types of Intent(Implicit and Explicit Intent)**

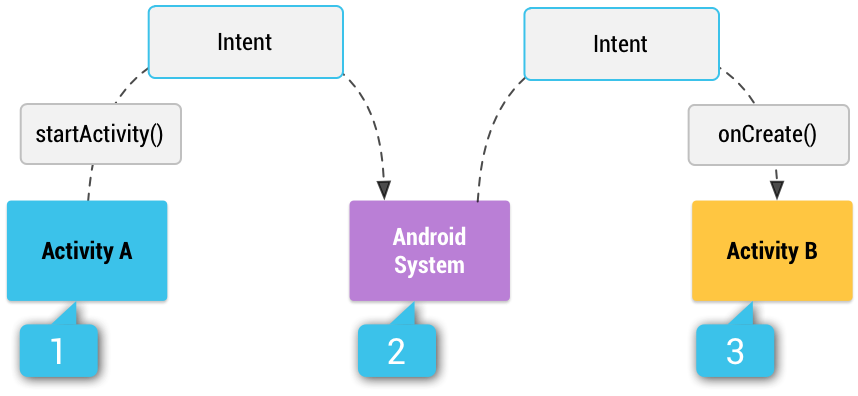
=> There are two types of Intents:

1. **Explicit Intents:** If you want communication between the components of your application only then you can use the Explicit Intents. Explicit Intents are used to communicate with a particular component of the same application. For example, if you want to launch an Activity by clicking some button on the present Activity then you can specify the fully-qualified address of the desired Activity to launch that Activity. Since this approach requires a fully-qualified address, you can use this approach in your own application i.e. you can use Explicit Intents to have communication in your own application.

Example: Shift from one activity to another activity

1. **Implicit Intents:** Here, you don’t need to specify the fully-qualified address. All you need to do is just specify the action that is to be performed by an Intent. By using the Implicit Intents you can communicate between various applications present in the mobile device.

Example: Start camera intent on button click



In the above figure, you can find that if you call an Explicit Intent i.e. you are passing the fully-qualified address of the action to be performed then, the system will directly launch that activity or will start performing the desired activity.

If you call the Implicit Intent then, the Android System will search for all the available components that can be used to start that activity. This process is done by comparing the contents of the intent with the content present in the ***intent-filter*s** declared in the ***AndroidManifest.xml*** file. If there is only one intent-filter that is compatible with the content of the intent then the Android system will start the desired component. But if there are a number of ***intent-filters*** that are compatible with the content of the Intent then the Android System will show you a list of applications that can be used to perform that particular action. For example, if you want to share some image from the Gallery, then you will get a number of choices like WhatsApp, Facebook, Instagram, Shareit, Gmail and many more image sharing applications. Now, you can choose any of the available choices.

Intent Filters are expressions that are used to specify the type of components or actions that can be received by the application and this Intent Filter is declared in the ***AndroiManifest.xml*** file. If you are not declaring any Intent Filters, then you have to use the Explicit Intents only.

#### **Information present in Intent**

So, we have seen that an Activity or an action can be called by using Explicit Intents or by using some Implicit Intents. But the question that arises here is that how does the Android System come to know that a particular Activity or Action is to be called? This is done by reading the information that is present in the Intent. The Android System reads the information present in the Intent and based on this information, the Android System decides which Activity is to be launched. So, some of the basic information that an Intent contains are:

1. **Action:** An action is a string that specifies the action to be performed by a particular Activity. For example, you can use the ***ACTION\_VIEW*** with ***startActivity()*** when your application contains some information like images that can be shown to the user. Another action that can be performed is ***ACTION\_SEND***, which is used to share some data with another application like in Email applications.
2. **Data:** While creating an Intent, you can pass the data and the type of data on which the action is to be performed by the Android System with the help of Intents. The URI object is used to reference the data that will be used to perform some action on it. For example, if you want to edit some data then you have to pass the URI of the data in your ***ACTION\_EDIT*** action.
3. **Category:** Category is used in case of Explicit Intents where you need to specify the type of application that will be used to perform a particular action. For example, if you want to send some data then only data sending applications should be made available for choice to the users. You can specify a category with the help of ***addCategory()***. Any number of categories can be added to the Intent.
4. **Component Name:** The component name is the name of the component that is to be started. You can set the component name by using ***setComponent()*** or ***setClass()*** or with the Intent Constructor.
5. **Extras:** You can add extra data to an Intent in the form of key-value pairs and this extra information can be passed from one Activity to the other. ***putExtra()*** is used to add some extra data to the Intents and this method accepts two parameters i.e. the key and its corresponding value.

**Examples for both:**

Refer: Github IntentDemo example

**2.5 Layout Manager**

=> The Android application having an excellent UI will have a large number of users because usually, people get attracted towards the look and feel of the application. So, to make the UI of the application good, we need to learn about XML(Extensible Markup Language) and View components in Android. So, in this blog, we will learn about User Interface(UI) in Android. Here, we are going to start with **XML** (extensible markup language) and design a screen and understand what the different types of View Components Android Studio provided to us. So, let's start with View.

**2.5.1 View and View Group:**

**View:** The view is the component which Android provides us to design the layouts of the app. So, we can understand the view as a rectangular area which is going to contain some element inside it. A *View* is a superclass for all the UI components.

examples:

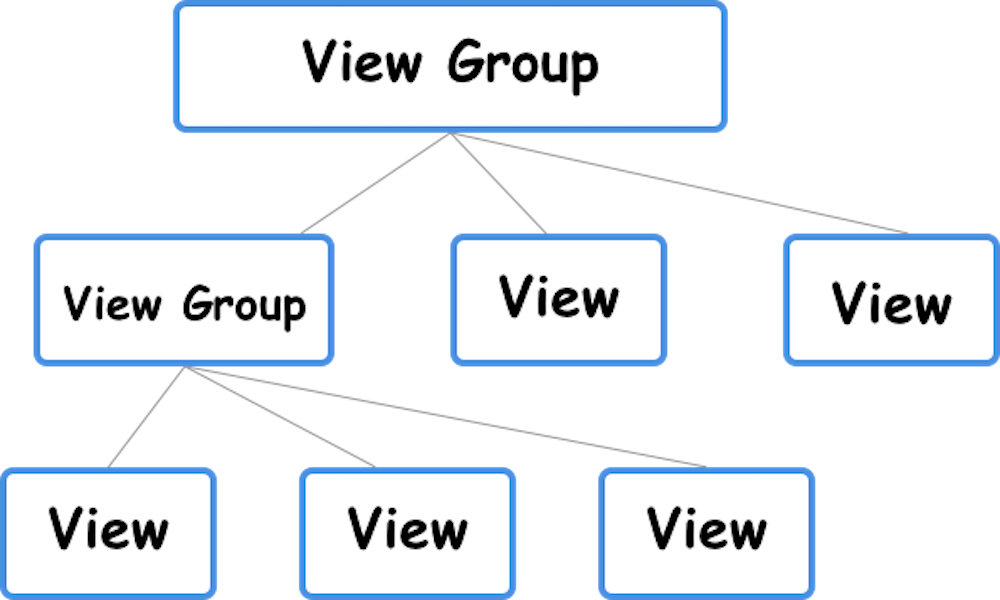
* **TextView:** To add some text in your application.
* **EditText:** This is used when you want to take some input from the users.
* **ImageView:** To add some image in the application.
* **ProgressBar:** To show the progress to something. For example, the loading screen.
* **Button:** Buttons are used to trigger some action on the click of the button. It can be starting a new activity or something else.
* **ImageButton:** It is used to make a clickable image.
* **CheckBox:** CheckBox is used to select some options out of many available options.
* **DatePicker:** To select some particular date.

**View Group:** A group of views is known as *ViewGroup*. The Top-level ViewGroup is a parent, and under it, all the view and other view groups are its children. *For example*, under a LinearLayout, you can add two Buttons and one EditText. Here, LinearLayout is the parent view and the Buttons and EditTexts are the children.

examples:

* **Linear Layout**
* **Relative Layout**
* **Constraint Layout**
* **Radio Group** etc

Diagram:



**2.5.2 Linear Layout:**

Linear means in a line, either horizontal or vertical. We can understand here as all the elements inside the linear layout get arranged linearly, one after the other. If you are using horizontal orientation, then all the views inside the LinearLayout will be arranged horizontally one after the other.

*For example:*  A Button is there on the screen, and now we need to put some text below the button. Here, we need to define the orientation of the LinearLayout to be vertical.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<Button

android:id="@+id/button2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Button" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="MindOrks" />

</LinearLayout>

**2.5.3 Relative Layout:**

Relative means *concerning another*, or we can understand this as a *relative to one another.* In this layout, all the components are arranged concerning each other.

***For example:*** If we have one *Button* on the screen and now we want to put the *TextView* below the button i.e. the *TextView* is relatively below the *Button*. Following is the code of doing the same:

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<Button

android:id="@+id/button2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Click Me!!" />

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_below="@id/button2"

android:text="KAILAS" />

</RelativeLayout>

**2.5.4 Table Layout:**

TableLayout positions its children into rows and columns. TableLayout containers do not display border lines for their rows, columns, or cells. The table will have as many columns as the row with the most cells. A table can leave cells empty. Cells can span multiple columns, as they can in HTML.

TableRow objects are the child views of a TableLayout (each TableRow defines a single row in the table). Each row has zero or more cells, each of which is defined by any kind of other View. So, the cells of a row may be composed of a variety of View objects, like ImageView or TextView objects. A cell may also be a ViewGroup object (for example, you can nest another TableLayout as a cell).

The following sample layout has two rows and two cells in each. The accompanying screenshot shows the result, with cell borders displayed as dotted lines (added for visual effect).

<?xml version="1.0" encoding="utf-8"?>

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:stretchColumns="1">

<TableRow>

<TextView

android:text="@string/table\_layout\_4\_open"

android:padding="3dip" />

<TextView

android:text="@string/table\_layout\_4\_open\_shortcut"

android:gravity="right"

android:padding="3dip" />

</TableRow>

<TableRow>

<TextView

android:text="@string/table\_layout\_4\_save"

android:padding="3dip" />

<TextView

android:text="@string/table\_layout\_4\_save\_shortcut"

android:gravity="right"

android:padding="3dip" />

</TableRow>

</TableLayout>

**2.5.5 Grid Layout:**

GridLayout is a type of android layout to create android applications which display widgets and text fields in grid format. Grid layout is usable with rows and columns like application developers can define how many rows and columns will be created in layout files. It's like creating a matrix layout in an android application. So here is the complete step by step tutorial for Android GridLayout Example Tutorial.

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:columnCount="4"

android:rowCount="3"

tools:context="com.android\_examples.com.gridlayout.MainActivity" >

<TextView

android:id="@+id/textView1"

android:layout\_row="0"

android:layout\_column="0"

android:text="A"

android:textAppearance="?android:attr/textAppearanceLarge"

android:padding="30dp"/>

<TextView

android:id="@+id/textView2"

android:layout\_row="0"

android:layout\_column="1"

android:text="B"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView3"

android:layout\_row="0"

android:layout\_column="2"

android:text="C"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView4"

android:layout\_row="0"

android:layout\_column="3"

android:text="D"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

//2nd row starts from here.

<TextView

android:id="@+id/textView5"

android:layout\_row="1"

android:layout\_column="0"

android:text="E"

android:textAppearance="?android:attr/textAppearanceLarge"

android:padding="30dp"/>

<TextView

android:id="@+id/textView6"

android:layout\_row="1"

android:layout\_column="1"

android:text="F"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView7"

android:layout\_row="1"

android:layout\_column="2"

android:text="G"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView8"

android:layout\_row="1"

android:layout\_column="3"

android:text="H"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

//3rd row starts from here.

<TextView

android:id="@+id/textView9"

android:layout\_row="2"

android:layout\_column="0"

android:text="I"

android:textAppearance="?android:attr/textAppearanceLarge"

android:padding="30dp"/>

<TextView

android:id="@+id/textView10"

android:layout\_row="2"

android:layout\_column="1"

android:text="J"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView11"

android:layout\_row="2"

android:layout\_column="2"

android:text="K"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

<TextView

android:id="@+id/textView12"

android:layout\_row="2"

android:layout\_column="3"

android:text="L"

android:padding="30dp"

android:textAppearance="?android:attr/textAppearanceLarge" />

</GridLayout>

**2.5.6 Constraint Layout:**

It makes handling complex screen designs easier. It also improves the performance of complex layouts.

With ConstraintLayout, even complex screen designs with nested layouts can be fast.

ConstraintLayout provides a level of flexibility that allows many of the features of older layouts to be achieved with a single layout instance. Before, you needed to nest multiple layouts.

This has the benefit of avoiding many problems inherent in nesting layouts. It allows designing so-called *flat* or *shallow* layout hierarchies. This leads to less complex layouts and improved user interface rendering performance at runtime.

ConstraintLayout is also implemented with an eye toward addressing the wide range of Android device screen sizes available on the market today.

The flexibility of ConstraintLayout makes it easier for user interfaces to be designed that respond and adapt to the device on which the app is running.

**2.5.7 Frame Layout:**

Frame Layout is one of the layouts which helps us to create a more complex design easily. When we are required to create a design where the components are on top of each other, we use the FrameLayout.

To define which component will be on top, we put it in the end. *For example*, if we want some text over an image, then we will put the TextView in the end.

<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:src="@drawable/mindorkslogo"

android:text="Button" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="MindOrks" />

</FrameLayout>

**2.5.8 Scroll Layout (ScrollView):**

A view group that allows the view hierarchy placed within it to be scrolled. Scroll view may have only one direct child placed within it. To add multiple views within the scroll view, make the direct child you add a view group, for example LinearLayout, and place additional views within that LinearLayout.

Scroll view supports vertical scrolling only. For horizontal scrolling, use [HorizontalScrollView](https://developer.android.com/reference/android/widget/HorizontalScrollView) instead.

Never add a [RecyclerView](https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html) or [ListView](https://developer.android.com/reference/android/widget/ListView) to a scroll view. Doing so results in poor user interface performance and a poor user experience.

<?xml version="1.0" encoding="utf-8"?>

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:fillViewport="false">

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical" android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView android:id="@+id/loginscrn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="80dp"

android:text="ScrollView"

android:textSize="25dp"

android:textStyle="bold"

android:layout\_gravity="center"/>

<TextView android:id="@+id/fstTxt"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="20dp"

android:text="Welcome to Tutlane"

android:layout\_gravity="center"/>

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button One" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Two" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Three" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Four" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Five" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Six" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Seven" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Eight" />

<Button android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="60dp"

android:text="Button Nine" />

</LinearLayout>

</ScrollView>

**Unit 3: BASIC UI DESIGN**

3.1 Button, Check Box, Radio Button, Toggle Button, Image Button

3.2 Text Fields(Edit Text)

3.3 Spinner

3.4 List View

3.5 Toast

3.6 Scroll View

3.7 Progress Bar

3.8 Auto Complete TextView

3.9 Dialog Box

4.0 Alert Dialog

4.1 Date Picker Dialog

4.2 Time Picker Dialog

4.3 Custom Dialog

**Unit 4: ADAPTER AND MENU**

4.1 Base Adapter

4.2 Array Adapter

4.3 ListView using Adapter

4.4 GridView using Adapter

4.5 Photo Gallery using Adapter

4.6 Menu (Option Menu, Context Menu, Popup Menu)

**Unit 5: THREADS AND NOTIFICATION**

5.1 Worker Thread

5.2 Handlers & Runnable

5.3 AsyncTask

5.4 Broadcast Receiver

5.5 Services

5.6 Service Life Cycle

5.7 Bounded Service

5.8 Unbounded Service

5.9 Notification

6.0 Alarm

6.1 Accessing Phone Services (Call, Sms)

**Unit 6: CONTENT PROVIDER**

6.1 Content Providers

6.2 SQLite Programming

6.3 SQLiteOpenHelper

6.4 SQLite Database

6.5 Cursor

6.6 Searching for content

6.7 Adding, Changing and removing content

6.8 Binding and executing queries

6.9 Android JSON

**Unit 7: LOCATION BASED SERVICES AND GOOGLE MAP**

7.1 Display Google Maps

7.2 Creating the Project

7.3 Obtaining the Maps API Key

7.4 Displaying the Map

7.5 Displaying the Zoom Control

7.6 Changing Views

7.7 Navigating to a specific location

7.8 Adding Markers

7.9 Getting the location that was touched

8.0 Geocoding and Reverse Geocoding

8.1 Getting Location Data

8.2 Monitoring a Location