Savitribai Phule Pune University Faculty of Information Technology

414467: Computer Laboratory – X
BEIT (2015 Course)

Semester - II

Teaching Scheme		Examination Scheme	
Practical :	2 Hrs. / Week	Term work :	25 Marks
		Oral	25 Marks

LABORATORY MANUAL

ACADEMIC YEAR 2020-21

PROGRAM OUTCOMES

The students in the Information Technology course will attain:

a. An ability to apply knowledge of computing, mathematics including discrete mathematics as well as probability and statistics, science, engineering and technology.

b. An ability to define a problem and provide a systematic solution with the help of conducting experiments, as well as analyzing and interpreting the data.

c. An ability to design, implement, and evaluate a software or a software/hardware co-system, component, or process to meet desired needs within realistic constraints.

d. An ability to identify, formulate, and provide systematic solutions to complex engineering problems.

e. An ability to use the techniques, skills, and modern engineering technologies tools, standard processes necessary for practice as an IT professional.

f. An ability to apply mathematical foundations, algorithmic principles, and Information Technology theory in the modeling and design of computer-based systems with necessary constraints and assumptions.

g. An ability to analyze the local and global impact of computing on individuals, organizations and society.

h. An ability to understand professional, ethical, legal, security and social issues and responsibilities.

i. An ability to function effectively as an individual or as a team member to accomplish a desired goal(s).

j. An ability to engage in life-long learning and continuing professional development to cope up with fast changes in the technologies/tools with the help of electives, professional organizations and extra-curricular activities.

k. An ability to communicate effectively in engineering community at large by means of effective presentations, report writing, paper publications, demonstrations.

l. An ability to understand engineering, management, financial aspects, performance, optimizations and time complexity necessary for professional practice.

m. An ability to apply design and development principles in the construction of software systems of varying complexity.

SCTR's Pune Institute of Computer Technology, Pune Department of Information Technology

Compliance
Document Control

Reference Code	**PICT-IT / Lab Manual Procedures**
Version No	2.0
Compliance Status	Complete
Revision Date	18 Feb 2021
Academic Year	2020-21
Security Classification	Department Specific
Document Status	Definitive
Review Period	Yearly

Authors

Name of Faculty	**Name of Institute**
Prof. R. R. Chhajed	PICT, Pune
Prof. A. C. Karve	PICT, Pune

Savitribai Phule Pune University
FACULTY OF INFORMATION TECHNOLOGY

Syllabus

414467: COMPUTER LABORATORY-X
(2015 Course)

Teaching Scheme:	**Examination Scheme:**
Practical: 2 Hours/Week	Term-work: 25 Marks Practical: 25 Marks

Credit: 01

Prerequisites:

1. Computer Network Technology.
2. Human Computer Interface.

Course Objectives:

1. To design and implement user interfaces for performing database operations.
2. To design applications for accessing smart devices and data generated through sensors and services.
3. To implement authentication protocols for providing security.

Course Outcomes:

Upon successful completion of this course student will be able to

1. Set up the Android environment and explain the Evolution of cellular networks.
2. Develop the User Interfaces using pre-built Android UI components.
3. Create applications for performing CURD SQLite database operations using Android.
4. Create the smart android applications using the data captured through sensors.
5. Implement the authentication protocols between two mobile devices for providing. Security.
6. Analyze the data collected through android sensors using any machine learning algorithm.

Guidelines:

This Computer Laboratory-X course has ubiquitous computing as a core subject. The problem statements should framed based on first six assignments mentioned in the syllabus. The teachers will frame the problem statements with due consideration that students have three hours to complete that. The practical examination will comprise of implementation and related theory. All assignments to be performed in Java 9.

Tools Required Android SDK / Android Studio, SQL Lite, Sensors, Arduino kit.

Laboratory Assignments

1. Android development environment. Installing and setting up the environment. Hello world application. Running the emulator. Inserting debug messages.

2. Android UI Design: Design a User Interface using pre-built UI components such as structured layout objects, UI controls and special interfaces such as dialogs, notifications, and menus. Also make this UI attractive using Android graphics platform OpenGL.

3. Android-database Connectivity: Create a SQLite Database for an Android Application and perform CRUD (Create, Read, Update and Delete) database operations.

4. Develop a Smart Light System (Light that automatically switched on in evening and gets off in morning) using open source Hardware platform like Arduino and some sensors (Light dependent resistor) and actuator (An LED)

5. Design and Develop a GUI for FAN regulator that uses Android platform.

6. Develop an Android based FAN regulator using open source Hardware platform like NodeMcu and actuator (a SERVO Motor).

7. Wireless Network: Develop an app for a rolling display program of news on computer display. The input strings are supplied by the mobile phone/ by another computer connected through wireless networks.

8. Android and Machine Learning: Healthcare System
fetching data from android and analysis using machine Learning.(Healthcare, Agricultural or smart city)

9. Case Study: Evolution of cellular networks all the way up to 7G.

10 App or web based Mini project on recent trends in Ubiquitous computing. (Like Smart City, Health, Agricultural etc.)

Links for Laboratory Assignments

1. https://developer.android.com/
2. https://www.androidhive.info/2011/11/android-sqlite-database-tutorial/
3. https://developers.google.com/android/guides/api-client
4. https://developer.android.com/guide/topics/sensors/sensors_overview

Lab Planning (Scheduling)

Sr. No.	Title	No. of Hrs.	Week
1	Android development environment. Installing and setting up the environment. Hello world application. Running the emulator. Inserting debug messages.	2	1
2	Android UI Design: Design a User Interface using pre-built UI components such as structured layout objects, UI controls and special interfaces such as dialogs, notifications, and menus. Also make this UI attractive using Android graphics platform OpenGL.	2	2

3	Android-database Connectivity: Create a SQLite Database for an Android Application and perform CRUD (Create, Read, Update and Delete) database operations.	2	3
4	Develop a Smart Light System (Light that automatically switched on in evening and gets off in morning) using open source Hardware platform like Arduino and some sensors (Light dependent resistor) and actuator (An LED)	2	4
5	Design and Develop a GUI for FAN regulator that uses Android platform.	2	5
6	Develop an Android based FAN regulator using open source Hardware platform like NodeMcu and actuator (a SERVO Motor).	2	6
7	Wireless Network: Develop an app for a rolling display program of news on computer display. The input strings are supplied by the mobile phone/ by another computer connected through wireless networks.	2	7
8	Android and Machine Learning: Healthcare System fetching data from android and analysis using machine Learning.(Healthcare, Agricultural or smart city)	2	8
9	Case Study: Evolution of cellular networks all the way up to 7G.	2	9

10	App or web based Mini project on recent trends in Ubiquitous computing. (Like Smart City, Health, Agricultural etc.)	2	10

Assignment No-1

Aim: Android development environment. Installing and setting up the environment. Hello world application. Running the emulator. Inserting debug messages.

Objective:

- Install and use the Android IDE.
- Understand the development process for building Android apps.
- Create an Android project from a basic app template.

Theory:

Android Studio is Google's IDE for Android apps. Android Studio gives you an advanced code editor and a set of app templates. In addition, it contains tools for development, debugging, testing, and performance that make it faster and easier to develop apps. You can test your apps with a large range of preconfigured emulators or on your own mobile device, and build production APKs for publication.

To get up and running with Android Studio:

- You may need to install the Java Development Kit - Java 7 or better.
- Install Android Studio

Task 1. Install Android Studio

Android Studio is available for Windows, Mac, and Linux computers. The installation is similar for all platforms. Any differences will be noted in the sections below.

Installing the Java Development Kit

1. On your computer, open a terminal window.
2. Type java -version

The output includes a line:

```
Java (™) SE Runtime Environment (build1.  X.0_05-b13)
```

X is the version number to look at.

-If this is 7 or greater, you can move on to installing Android Studio.

-If you see a Java SE version is below 7 or if Java is not installed, you need to install the latest version of the Java SE development kit before installing Android Studio.

Installing Android Studio

1. Navigate to the Android developers site and follow the instructions to download and install Android Studio.
    - Accept the default configurations for all steps.
    - Make sure that all components are selected for installation.
2. After finishing the install, the Setup Wizard will download and install some additional components.
3. When the download completes, Android Studio will start, and you are ready to create your first project.

Task 2: Create "Hello World" app

In this task, you will implement the "Hello World" app to verify that Android studio is correctly installed and learn the basics of developing with Android Studio.

2.1 Create the "Hello World" app

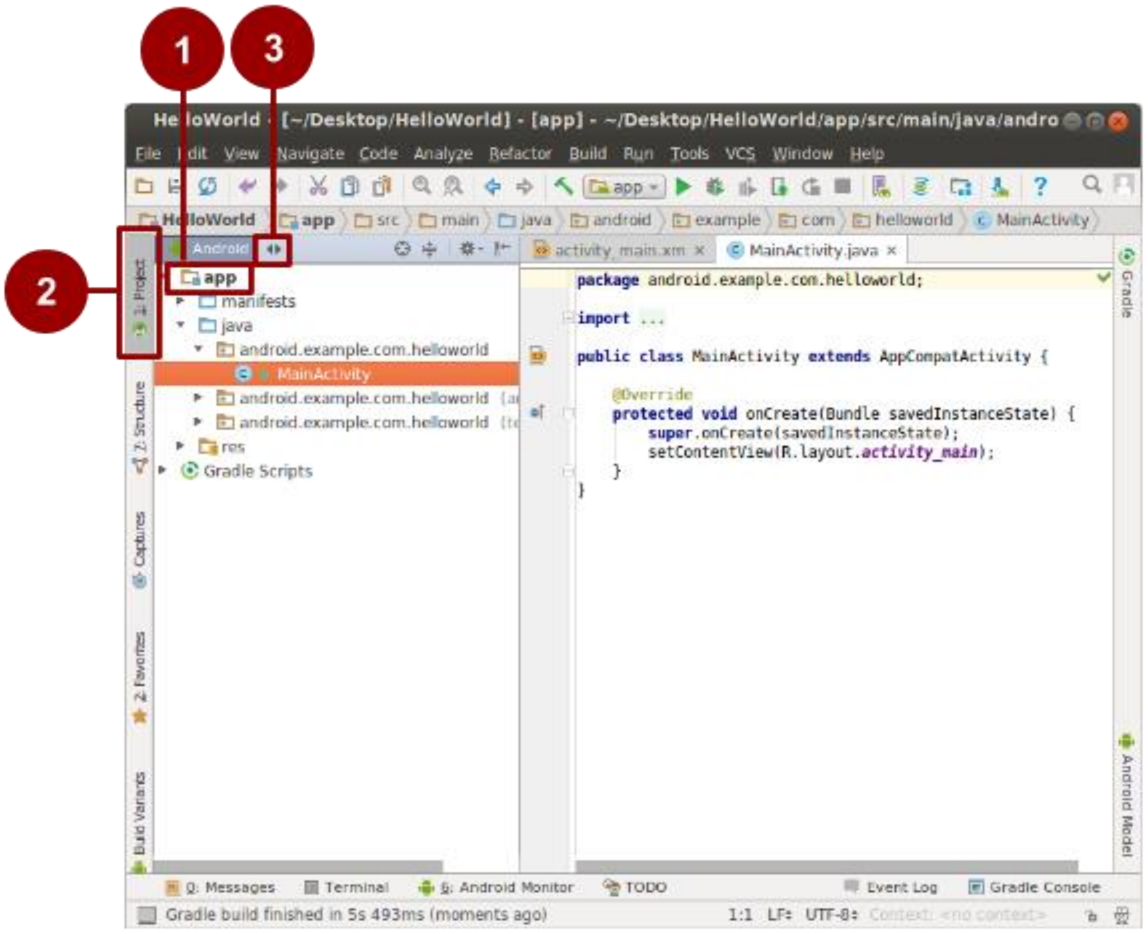
1. Launch Android Studio.
2. In the main **Welcome to Android Studio** window, click "Start a new Android Studio project".
3. In the **New Project** window, give your application an **Application Name**, such as "Hello World".
4. Verify the Project location, or choose a different directory for storing your project.
5. Choose a unique **Company Domain**.
    - Apps published to the Google Play Store must have a unique package name. Since domains are unique, prepending your app's name with your or your company's domain name is going to result in a unique package name.
    - If you are not planning to publish your app, you can accept the default example domain. Be aware that changing the package name of your app later is extra work.
6. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory. Click Next.
7. On the **Target Android Devices** screen, "Phone and Tablet" should be selected.
8. Click **Next**.

9. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically. Click **Next**.
10. **Customize the Activity** window. Every app needs at least one activity. An activity represents a single screen with a user interface and Android Studio provides templates to help you get started. For the Hello World project, choose the simplest template (as of this writing, the "Empty Activity" project template is the simplest template) available.
11. It is a common practice to call your main activity MainActivity. This is not a requirement.
12. Make sure the **Generate Layout file** box is checked (if visible).
13. Make sure the **Backwards Compatibility (App Compat)** box is checked.
14. Leave the **Layout Name** as activity_main. It is customary to name layouts after the activity they belong to. Accept the defaults and click **Finish**.

After these steps, Android Studio:

- Creates a folder for your Android Studio Projects.
- Builds your project with Gradle (this may take a few moments). Android Studio uses Gradle as its build system. See the Configure your build developer page for more information.
- Opens the code editor with your project.
- Displays a tip of the day.
  - Android Studio offers many keyboard shortcuts, and reading the tips is a great way to learn them over time.

The Android Studio window should look similar to the following diagram:


1
3
2
HelloWorld - [~/Desktop/HelloWorld] - [app] - ~/Desktop/HelloWorld/app/src/main/java/andro
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloWorld app src main java android example com helloworld MainActivity
Android
1: Project
7: Structure
Captures
2: Favorites
Build Variants
app
manifests
java
android.example.com.helloworld
MainActivity
android.example.com.helloworld
android.example.com.helloworld
res
Gradle Scripts
activity_main.xml
MainActivity.java
Gradle
Android Model
0: Messages
Terminal
6: Android Monitor
TODO
Event Log
Gradle Console
1:1 LF UTF-8 Context: <no context>

```
package android.example.com.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Gradle build finished in 5s 493ms (moments ago)

You can look at the hierarchy of the files for your app in multiple ways.

1. Click on the Hello World folder to expand the hierarchy of files (1),
2. Click on **Project (2)**.
3. Click on the **Android** menu (3).
4. Explore the different view options for your project.

Task 3: Explore the project structure (Optional)

In this practical, you will explore how the project files are organized in Android Studio.

These steps assume that your Hello World project starts out as shown in the diagram above.

3.1 Explore the project structure and layout

In the Project > Android view of your previous task, there are three top-level folders below your **app**folder: **manifests**, **java**, and **res**.

1. Expand the **manifests** folder.

   This folder contains **AndroidManifest.xml.** This file describes all of the components of your Android app and is read by the Android run-time system when your program is executed.

2. Expand the **java** folder. All your Java language files are organized in this folder. The **java** folder contains three subfolders:

- **com.example.hello.helloworld (or the domain name you have specified):** All the files for a package are in a folder named after the package. For your Hello World application, there is one package and it only contains MainActivity.java (the file extension may be omitted in the Project view).
- **com.example.hello.helloworld(androidTest):** This folder is for your instrumented tests, and starts out with a skeleton test file.
- **com.example.hello.helloworld(test):** This folder is for your unit tests and starts out with an automatically created skeleton unit test file.

3. Expand the **res** folder. This folder contains all the resources for your app, including images, layout files, strings, icons, and styling. It includes these subfolders:

- **drawable**. Store all your app's images in this folder.
- **layout**. Every activity has at least one layout file that describes the UI in XML. For Hello World, this folder contains activity_main.xml.
- **mipmap**. Store your launcher icons in this folder. There is a sub-folder for each supported screen density. Android uses the screen density, that is, the number of pixels per inch to determine the required image resolution. Android groups all actual screen densities into generalized densities, such as medium (mdpi), high (hdpi), or extra-extra-extra-high (xxxhdpi). The ic_launcher.png folder contains the default launcher icons for all the densities supported by your app.
- **values**. Instead of hardcoding values like strings, dimensions, and colors in your XML and Java files, it is best practice to define them in their respective values file. This makes it easier to change and be consistent across your app.

4. Expand the **values** subfolder within the res folder. It includes these subfolders:

- **colors.xml**. Shows the default colors for your chosen theme, and you can add your own colors or change them based on your app's requirements.
- **dimens.xml**. Store the sizes of views and objects for different resolutions.
- **strings.xml**. Create resources for all your strings. This makes it easy to translate them to other languages.
- **styles.xml**. All the styles for your app and theme go here. Styles help give your app a consistent look for all UI elements.

3.2 The Gradle build system

Android Studio uses Gradle as its build system. As you progress through these practicals, you will learn more about gradle and what you need to build and run your apps.

1. Expand the **Gradle Scripts** folder. This folder contains all the files needed by the build system.
2. Look for the **build.gradle(Module:app)** file. When you are adding app-specific dependencies, such as using additional libraries, they go into this file.

Task 4: Create a virtual device (emulator)

In this task, you will use the Android Virtual Device (AVD) manager to create a virtual device or emulator that simulates the configuration for a particular type of Android device.

Using the AVD Manager, you define the hardware characteristics of a device and its API level, and save it as a virtual device configuration.

When you start the Android emulator, it reads a specified configuration and creates an emulated device that behaves exactly like a physical version of that device , but it resides on your computer .

Why: With virtual devices, you can test your apps on different devices (tablets, phones) with different API levels to make sure it looks good and works for most users. You do not need to depend on having a physical device available for app development.

4.1 Create a virtual device

In order to run an emulator on your computer, you have to create a configuration that describes the virtual device.

1. In Android Studio, select Tools > Android > AVD Manager, or click the AVD Manager icon in the toolbar.
2. Click the **+Create Virtual Device**…. (If you have created a virtual device before, the window shows all of your existing devices and the button is at the bottom.)

   The Select Hardware screen appears showing a list of preconfigured hardware devices. For each device, the table shows its diagonal display size (Size), screen resolution in pixels (Resolution), and pixel density (Density).

   For the Nexus 5 device, the pixel density is xxhdpi, which means your app uses the launcher icons in the xxhdpi folder of the mipmap folder. Likewise, your app will use layouts and drawables from folders defined for that density as well.

3. Choose the Nexus 5 hardware device and click **Next**.

4. On the **System Image** screen, from the **recommended** tab, choose which version of the Android system to run on the virtual device. You can select the latest system image.

   There are many more versions available than shown in the **recommended** tab. Look at the **x86 Images** and **Other Images** tabs to see them.

5. If a **Download** link is visible next to a system image version, it is not installed yet, and you need to download it. If necessary, click the link to start the download, and click **Finish** when it's done.
6. On **System Image** screen, choose a system image and click **next**.
7. Verify your configuration, and click **Finish**. (If the **Your Android Devices** AVD Manager window stays open, you can go ahead and close it.)

Task 5. Run your app on an emulator

In this task, you will finally run your Hello World app.

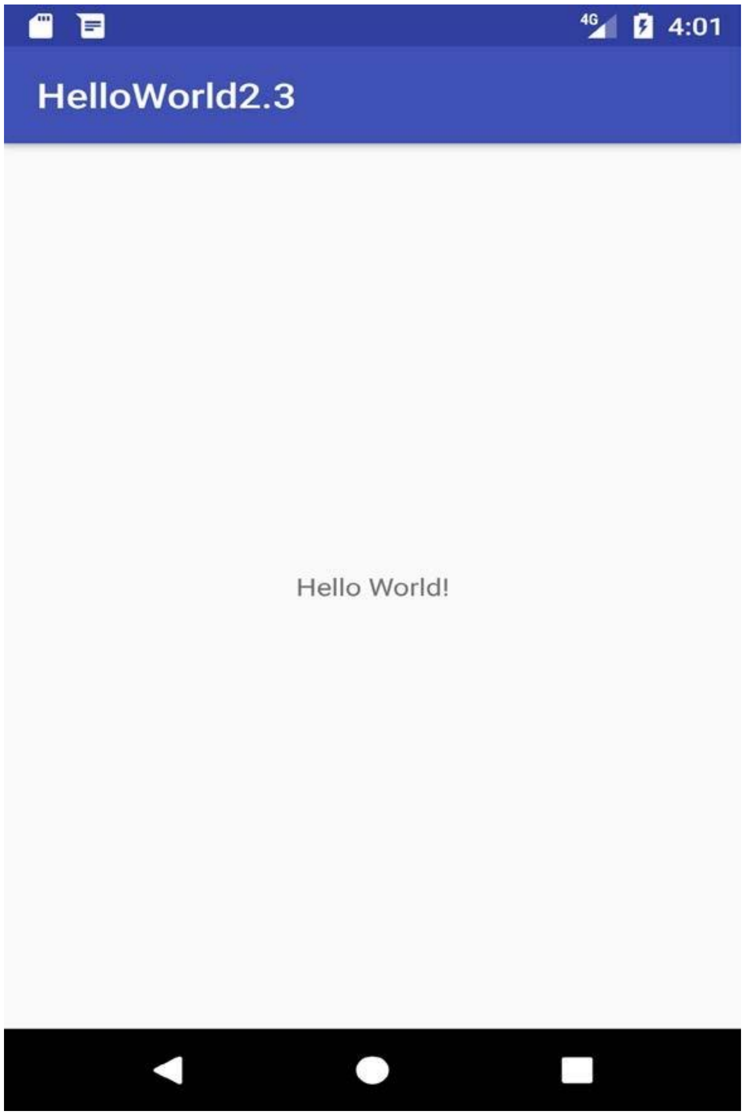
5.1 Run your app on an emulator

1. In Android Studio, select **Run** > **Run app** or click the **Run icon** ▶ in the toolbar.
2. In the **Select Deployment Target** window, under **Available Emulators**, select **Nexus 5 API 23** and click **OK**.

   The emulator starts and boots just like a physical device. Depending on the speed of your computer, this may take a while. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

   You should see the Hello World app as shown in the following screenshot.

OUTPUT



Hello World!

Note: When testing on an emulator, it is a good practice to start it up once, at the very beginning of your session. You should not close the emulator until you are done testing your app, so that your app doesn't have to go through the boot process again.

Task 6. Add log statements to your app

In this practical, you will add log statements to your app, which are displayed in the logging window of the Android Monitor.

Why: Log messages are a powerful debugging tool that you can use to check on values, execution paths, and report exceptions.

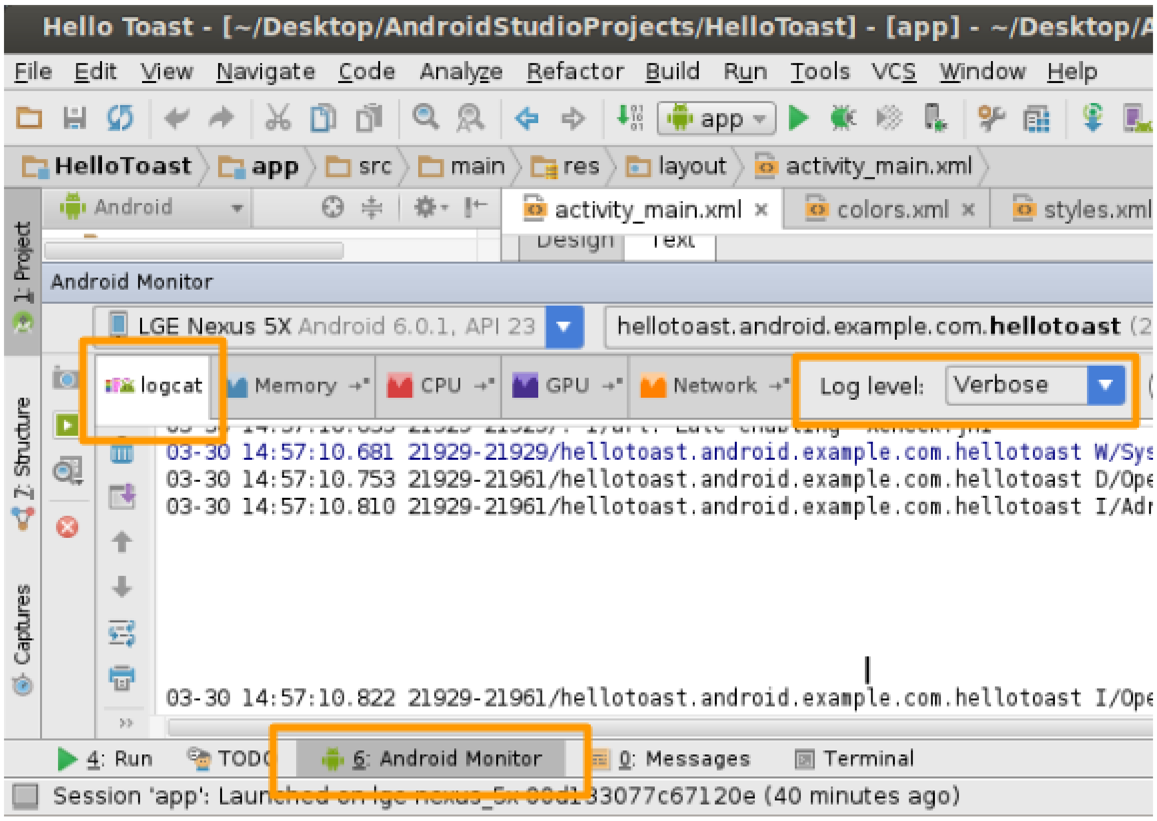
The **Android Monitor** displays information about your app.

1. Click the **Android Monito**r button at the bottom of Android Studio to open the Android Monitor.

   By default, this opens to the **logcat** tab, which displays information about your app as it is running. If you add log statements to your app, they are printed here as well.

   You can also monitor the Memory, CPU, GPU, and Network performance of your app from the other tabs of the Android Monitor. This can be helpful for debugging and performance tuning your code.

2. The default log level is **Verbose**. In the drop-down menu, change the log level to **Debug**.


Hello Toast - [~/Desktop/AndroidStudioProjects/HelloToast] - [app] - ~/Desktop/A
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloToast app src main res layout activity_main.xml
Android
activity_main.xml colors.xml styles.xml
Design Text
Android Monitor
LGE Nexus 5X Android 6.0.1, API 23
hellotoast.android.example.com.hellotoast (2
logcat Memory CPU GPU Network
Log level: Verbose
1: Project
7: Structure
Captures
4: Run TODO 6: Android Monitor 0: Messages Terminal

```
03-30 14:57:10.681 21929-21929/hellotoast.android.example.com.hellotoast W/Sys
03-30 14:57:10.753 21929-21961/hellotoast.android.example.com.hellotoast D/Ope
03-30 14:57:10.810 21929-21961/hellotoast.android.example.com.hellotoast I/Adr
03-30 14:57:10.822 21929-21961/hellotoast.android.example.com.hellotoast I/Ope
```

Session 'app': Launched on lge-nexus_5x-00d133077c67120e (40 minutes ago)

Log statements that you add to your app code print a message specified by you in the logcat tab of the Android Monitor. For example:

Log.d("MainActivity", "Hello World");

The parts of the message are:

- Log – The Log class. API for sending log messages.
- d – The Log level. Used to filter log message display in logcat. "d" is for debug. Other log levels are "e" for error, "w" for warning, and "i" for info.
- "MainActivity" – The first argument is a tag which can be used to filter messages in logcat. This is commonly the name of the activity from which the message originates. However, you can make this anything that is useful to you for debugging.

  By convention, log tags are defined as constants:

  private static final String LOG_TAG = MainActivity.class.getSimpleName();

- "Hello world" – The second argument is the actual message.

6.1 Add log statements to your app

1. Open your Hello World app in Android studio, and open MainActivity file.

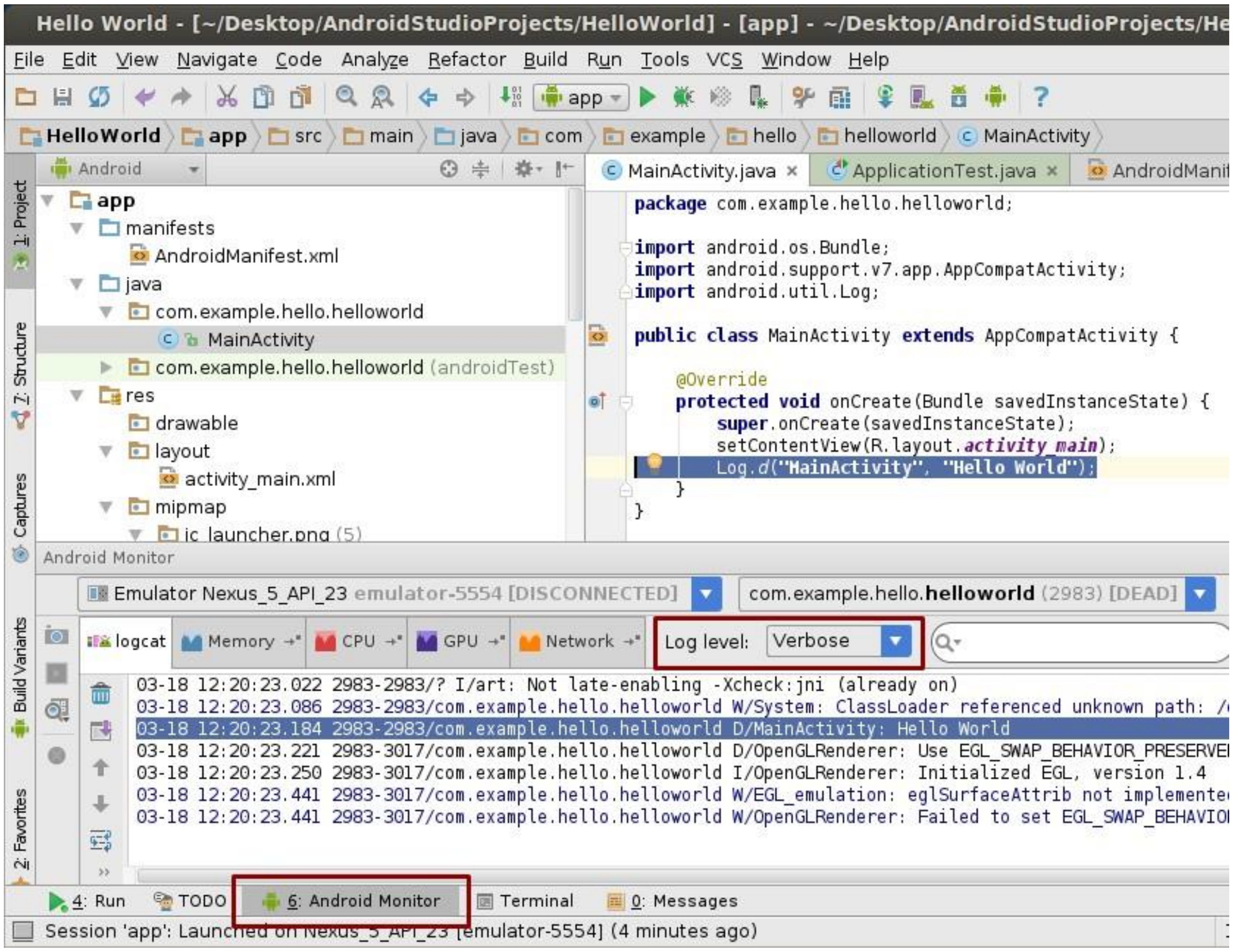
2. **File** > **Settings** > **Editor** > **General** >**Auto Import (**Mac: **Android Studio** > **Preferences** > **Editor** > **General** >**Auto Import)**. Select all check boxes and set **Insert imports on paste** to **All**. Unambiguous imports are now added automatically to your files. Note the "add unambiguous imports on the fly" option is important for some Android features such as NumberFormat. If not checked, NumberFormat shows an error. Click on 'Apply' followed by clicking on the 'Ok' button.
3. In the onCreate method, add the following log statement:
4. **Log.d("MainActivity", "Hello World");**
5. If the Android Monitor is not already open, click the Android Monitor tab at the bottom of Android Studio to open it. (See screenshot.)
6. Make sure that the Log level in the Android Monitor logcat is set to Debug or Verbose (default).
7. Run your app.

Solution Code:

```
package com.example.hello.helloworld;


import android.os.Bundle;
import
android.support.v7.app.AppCompatActivity;
import android.util.Log;


public class MainActivity extends
  AppCompatActivity { @Override
  protected void onCreate(Bundle
     savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     Log.d("MainActivity", "Hello World");
  }
```

Output Log Message

```
03-18 12:20:23.184 2983-2983/com.example.hello.helloworld D/MainActivity: Hello World
```


Hello World - [~/Desktop/AndroidStudioProjects/HelloWorld] - [app] - ~/Desktop/AndroidStudioProjects/He
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
HelloWorld app src main java com example hello helloworld MainActivity
Android
app
manifests
AndroidManifest.xml
java
com.example.hello.helloworld
MainActivity
com.example.hello.helloworld (androidTest)
res
drawable
layout
activity_main.xml
mipmap
ic_launcher.png (5)
MainActivity.java
ApplicationTest.java
AndroidManif
1: Project
7: Structure
Captures
Build Variants
2: Favorites
Android Monitor
Emulator Nexus_5_API_23 emulator-5554 [DISCONNECTED]
com.example.hello.helloworld (2983) [DEAD]
logcat
Memory
CPU
GPU
Network
Log level: Verbose
4: Run
TODO
6: Android Monitor
Terminal
0: Messages
Session 'app': Launched on Nexus_5_API_23 [emulator-5554] (4 minutes ago)

```
package com.example.hello.helloworld;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("MainActivity", "Hello World");
    }
}
```

```
03-18 12:20:23.022 2983-2983/? I/art: Not late-enabling -Xcheck:jni (already on)
03-18 12:20:23.086 2983-2983/com.example.hello.helloworld W/System: ClassLoader referenced unknown path: /
03-18 12:20:23.184 2983-2983/com.example.hello.helloworld D/MainActivity: Hello World
03-18 12:20:23.221 2983-3017/com.example.hello.helloworld D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVE
03-18 12:20:23.250 2983-3017/com.example.hello.helloworld I/OpenGLRenderer: Initialized EGL, version 1.4
03-18 12:20:23.441 2983-3017/com.example.hello.helloworld W/EGL_emulation: eglSurfaceAttrib not implemente
03-18 12:20:23.441 2983-3017/com.example.hello.helloworld W/OpenGLRenderer: Failed to set EGL_SWAP_BEHAVIO
```

Task 7: Explore the AndroidManifest.xml file

Every app includes an Android Manifest file (**AndroidManifest.xml**).The manifest file contains essential information about your app and presents this information to the Android runtime system. Android must have this information before it can run any of your app's code.
In this practical you will find and read the AndroidManifest.xml file for the Hello World app.

7.1 Explore the AndroidManifest.xml file

1. Open your Hello World app in Android studio, and in the **manifests** folder, open**AndroidManifest.xml**.
2. Read the file and consider what each line of code indicates. The code below is annotated to give you some hints.

Annotated code:

```
<!-- XML version and character encoding -->
<?xml version="1.0" encoding="utf-8"?>
<!-- Required starting tag for the manifest -->
<manifest
<!-- Defines the android namespace. Do not change. -->
xmlns:android="http://schemas.android.com/apk/res/android"
<!-- Unique package name of your app. Do not change once app is
    published. -->
  package="com.example.hello.helloworld">
 <!-- Required application tag -->
  <application
      <!-- Allow the application to be backed up and restored. –>
      android:allowBackup="true"
     <!-- Icon for the application as a  whole,
          and default icon for application components. –>
      android:icon="@mipmap/ic_launcher"
    <!-- User-readable for the application as a whole,
         and default icon for application components. Notice that Android
         Studio first shows the actual label "Hello World".
         Click on it, and you will see that the code actually refers to a string
         resource. Ctrl-click @string/app_name to see where the resource is
         specified. This will be covered in a later practical . –>
      android:label="@string/app_name"
    <!-- Whether the app is willing to support right-to-left layouts.–>
      android:supportsRtl="true"
    <!-- Default theme for styling all activities. –>
      android:theme="@style/AppTheme">
     <!-- Declares an activity. One is required.
          All activities must be declared,
          otherwise the system cannot see and run them. –>
      <activity
          <!-- Name of the class that implements the activity;
              subclass of Activity. –>
          android:name=".MainActivity">
          <!-- Specifies the intents that this activity can respond to.–>
          <intent-filter>
              <!-- The action and category together determine what
                  happens when the activity is launched. –>
              <!-- Start activity as the main entry point.
                  Does not receive data. –>
              <action android:name="android.intent.action.MAIN" />
              <!-- Start this activity as a top-level activity
                  in the launcher . –>
              <category android:name="android.intent.category.LAUNCHER" />
```

```
<!-- Closing tags –
                              >
      </intent-filter>
    </activity>
 </application>
```

Task 8. Explore the build.gradle file

Android Studio uses a build system called Gradle. Gradle does incremental builds, which allows for shorter edit-test cycles.

In this task, you will explore the **build.gradle** file.
Why: When you add new libraries to your Android project, you may also have to update your **build.gradle file**. It's useful to know where it is and its basic structure.

8.1 Explore the build.gradle(Module app) file

1. In your project hierarchy, find **Gradle Scripts** and expand it. There several build.gradle files. One with directives for your whole project, and one for each app module. The module for your app is called "app". In the Project view, it is represented by the **app** folder at the top- level of the Project view.
2. Open **build.gradle (Module.app).**
3. Read the file and learn what each line of code indicates.

Solution:

```
// Add Android-specific build tasks
apply plugin:
'com.android.application'
// Configure Android specific build
options. android {
   // Specify the target SDK version for the build.
  compileSdkVersion 23
  // The version of the build tools to use.
  buildToolsVersion "23.0.2"
  // Core settings and entries. Overrides manifest settings!
  defaultConfig {
     applicationId "com.example.hello.helloworld"
     minSdkVersion 15
     targetSdkVersion 23
     versionCode 1
     versionName "1.0"
  }
  // Controls how app is built and
  packaged. buildTypes {
     // Another common option is debug, which is not signed by default.
```

```
    release {
      // Code shrinker. Turn this on for production along with
      // shrinkResources.
      minifyEnabled false
      // Use ProGuard, a Java optimizer.
      proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
      rules.pro'
    }
  }
}
// This is the part you are most likely to change as you start using
// other libraries.
dependencies {
  // Local binary dependency. Include any JAR file inside
  app/libs. compile fileTree(dir: 'libs', include: ['*.jar'])
  // Configuration for unit tests.
  testCompile 'junit:junit:4.12'
  // Remote binary dependency. Specify Maven coordinates of the Support
  // Library needed. Use the SDK Manager to download and install such
  // packages.
  compile 'com.android.support:appcompat-v7:23.2.1'
}
```

Task 9. [Optional] Run your app on a device

In this final task, you will run your app on a physical mobile device such as a phone or tablet.

Why: Your users will run your app on physical devices. You should always test your apps on both virtual and physical devices.

What you need:

- An Android device such as a phone or tablet.
- A data cable to connect your Android device to your computer via the USB port.
- If you are using a Linux or Windows OS, you may need to perform additional steps to run on a hardware device. Check the Using Hardware Devices documentation. On Windows, you may need to install the appropriate USB driver for your device. See OEM USB Drivers.

9.1 [Optional] Run your app on a device

To let Android Studio, communicate with your device, you must turn on USB Debugging on your Android device. This is enabled in the Developer options settings of your device. Note this is not the same as rooting your device.

On Android 4.2 and higher, the Developer options screen is hidden by default. To show Developer options and enable USB Debugging:

1. On your device, open **Settings** > **About phone** and tap **Build number** seven times.
2. Return to the previous screen (**Settings**). **Developer options** appears at the bottom of the list. Click **Developer options**.
3. Choose **USB Debugging**.

Now you can connect your device and run the app from Android Studio.

1. Connect your device to your development machine with a USB cable.
2. In Android Studio, at the bottom of the window, click the Android Monitor tab. You should see your device listed in the top-left drop-down menu.
3. Click the **Run** button in the toolbar. The **Select Deployment Target** window opens with the list of available emulators and connected devices.
4. Select your device, and click **OK**.

Android Studio should install and runs the app on your device.

Conclusion:

- Thus we know how to Install and use the Android IDE. Also we understand the development process for building Android apps. We have created an Android project from a basic app template.

FAQs:-

What devices are supported for Google Play Instant?

Do developers need to build two different Android

apps? Can users choose to install the app permanently?

How do permissions work in Google Play

Instant?

Which permissions are available to an instant

app?

Lab. Assignment No – 2

Aim: Android UI Design: Design a User Interface using pre-built UI components such as structured layout objects, UI controls and special interfaces such as dialogs, notifications, and menus. Also make this UI attractive using Android graphics platform OpenGL.

Objective:

Theory:

Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

Layouts

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects, as shown in figure 1.

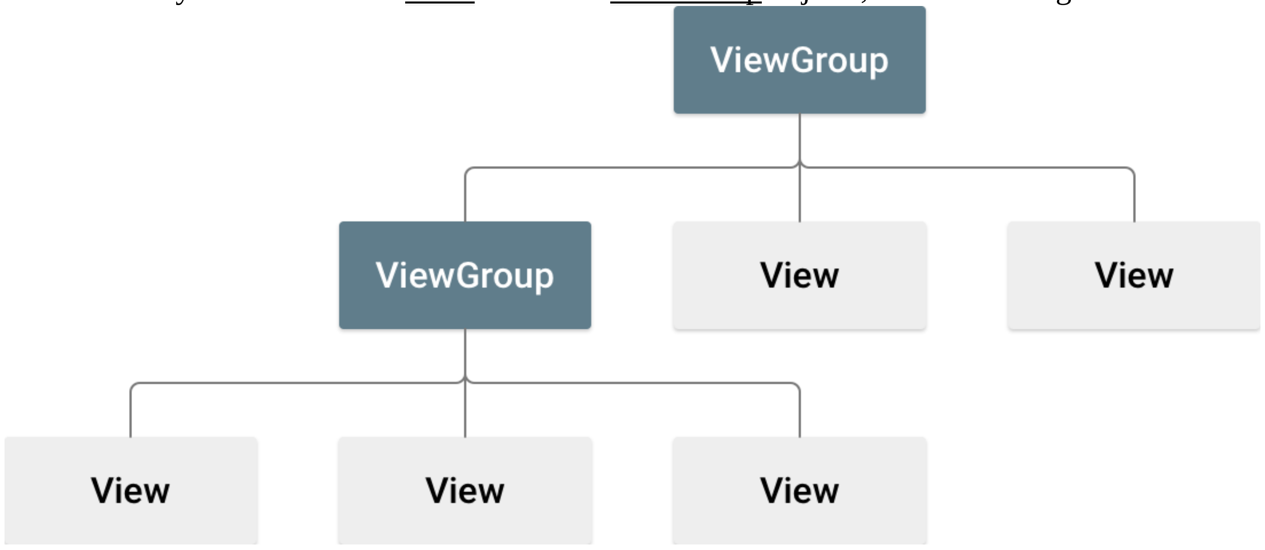


Figure 1. Illustration of a view hierarchy, which defines a UI layout

The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  You can also use Android Studios Layout Editor to build your XML layout using a drag-and-drop interface.

- **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Load the XML Resource

When you compile your app, each XML layout file is compiled into a View resource. You should load the layout resource from your app code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: **R.layout.***layout_file_name*. For example, if your XML layout is saved as **main_layout.xml**, you would load it for your Activity like so:

JAVA

```
public void onCreate(Bundle
  savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main_layout);
```

The **onCreate()** callback method in your Activity is called by the Android framework when your Activity is launched (see the discussion about lifecycles, in the Activities document).

Attributes

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the **textSize** attribute), but these attributes are also inherited by any View objects that may extend this class. Some are common to all View objects, because they are inherited from the root View class (like the **id**attribute). And, other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

ID

Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the app is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the **id** attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often. The syntax for an ID, inside an XML tag is:

```
android:id="@+id/my_button"
```

The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource. The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the **R.java**file). There are a number of other ID resources that are offered by the Android framework. When referencing an Android resource ID, you do not need the plus-symbol, but must add the **android** package namespace, like so:

```
android:id="@android:id/empty"
```

With the **android** package namespace in place, we're now referencing an ID from the **android.R** resources class, rather than the local resources class.

In order to create views and reference them from the app, a common pattern is to:

1. Define a view/widget in the layout file and assign it a unique ID:

```
<Button android:id="@+id/my_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/my_button_text"/
```

2. Then create an instance of the view object and capture it from the layout (typically in the onCreate() method):

JAVA

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Defining IDs for view objects is important when creating a RelativeLayout. In a relative layout, sibling views can define their layout relative to another sibling view, which is referenced by the unique ID.

An ID need not be unique throughout the entire tree, but it should be unique within the part of the tree you are searching (which may often be the entire tree, so it's best to be completely unique when possible).

Layout Parameters

XML layout attributes named **layout_*something*** define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As you can see in figure 2, the parent view group defines layout parameters for each child view (including the child view group).

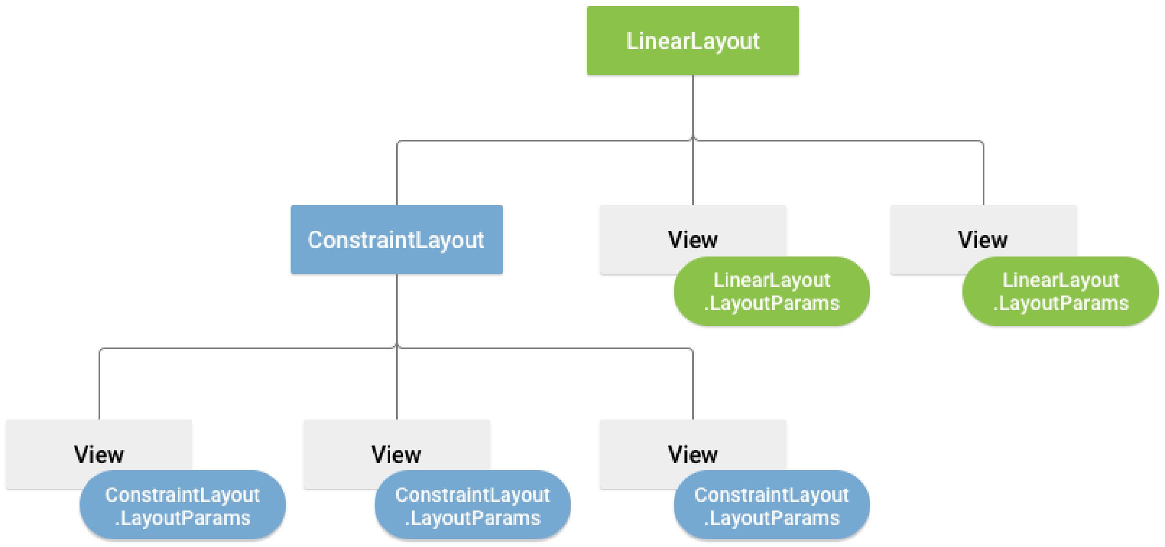

LinearLayout
ConstraintLayout
View
LinearLayout .LayoutParams
View
LinearLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams

Figure 2. Visualization of a view hierarchy with layout parameters associated with each view

Note that every LayoutParams subclass has its own syntax for setting values. Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children.

All view groups include a width and height (**layout_width** and **layout_height**), and each view is required to define them. Many LayoutParams also include optional margins and borders.

You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:

- ***wrap_content*** tells your view to size itself to the dimensions required by its content.
- ***match_parent*** tells your view to become as big as its parent view group will allow.

In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using relative measurements such as density-independent pixel units (***dp***), ***wrap_content***, or ***match_parent***, is a better approach, because it helps ensure that your app will display properly across a variety of device screen sizes. The accepted measurement types are defined in the Available Resources document.

Layout Position

The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of *left*and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.

It is possible to retrieve the location of a view by invoking the methods getLeft() and getTop(). The former returns the left, or X, coordinate of the rectangle representing the view. The latter returns the top, or Y, coordinate of the rectangle representing the view. These methods both return the location of the view relative to its parent. For instance, when **getLeft()** returns 20, that means the view is located 20 pixels to the right of the left edge of its direct parent.

In addition, several convenience methods are offered to avoid unnecessary computations, namely getRight() and getBottom(). These methods return the coordinates of the right and bottom edges of the rectangle representing the view. For instance, calling getRight() is similar to the following computation: **getLeft() + getWidth()**.

Size, Padding and Margins

The size of a view is expressed with a width and a height. A view actually possesses two pairs of width and height values.

The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent. The measured dimensions can be obtained by calling getMeasuredWidth() and getMeasuredHeight().

The second pair is simply known as *width* and *height*, or sometimes *drawing width* and *drawing height*. These dimensions define the actual size of the view on screen, at drawing time and after layout. These values may, but do not have to, be different from the measured width and height. The width and height can be obtained by calling getWidth() and getHeight().

To measure its dimensions, a view takes into account its padding. The padding is expressed in pixels for the left, top, right and bottom parts of the view. Padding can be used to offset the content of the view by a specific number of pixels. For instance, a left padding of 2 will push the view's content by 2 pixels to the right of the left edge. Padding can be set using the setPadding(int, int, int, int) method and queried by callin getPaddingLeft getPaddingTop and getPaddingBottom().
g

getPaddingRigh

Even though a view can define a padding, it does not provide any support for margins. However, view groups provide such a support. Refer
to ViewGroup andViewGroup.MarginLayoutParams for further information.

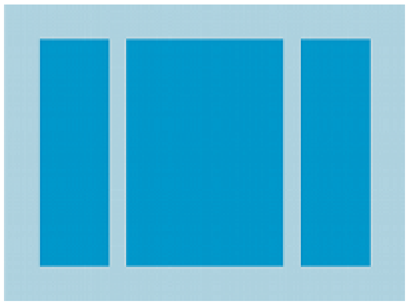
For more information about dimensions, see Dimension Values.

Common Layouts

Each subclass of the ViewGroup class provides a unique way to display the views you nest within it. Below are some of the more common layout types that are built into the Android platform.

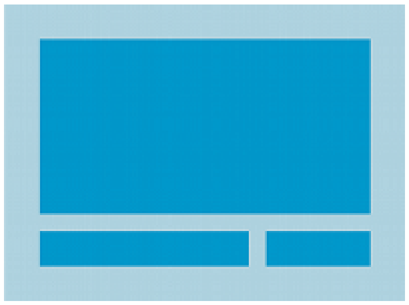
Note: Although you can nest one or more layouts within another layout to achieve your UI design, you should strive to keep your layout hierarchy as shallow as possible. Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a

Linear Layout



A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout



Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

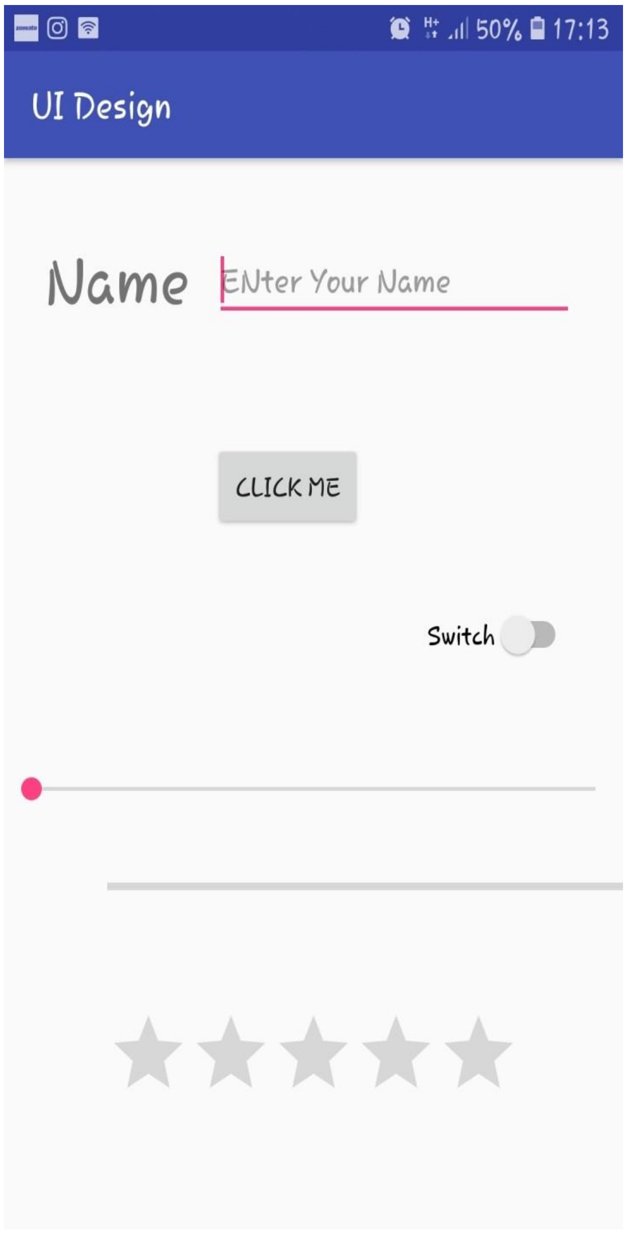
Displays web pages.

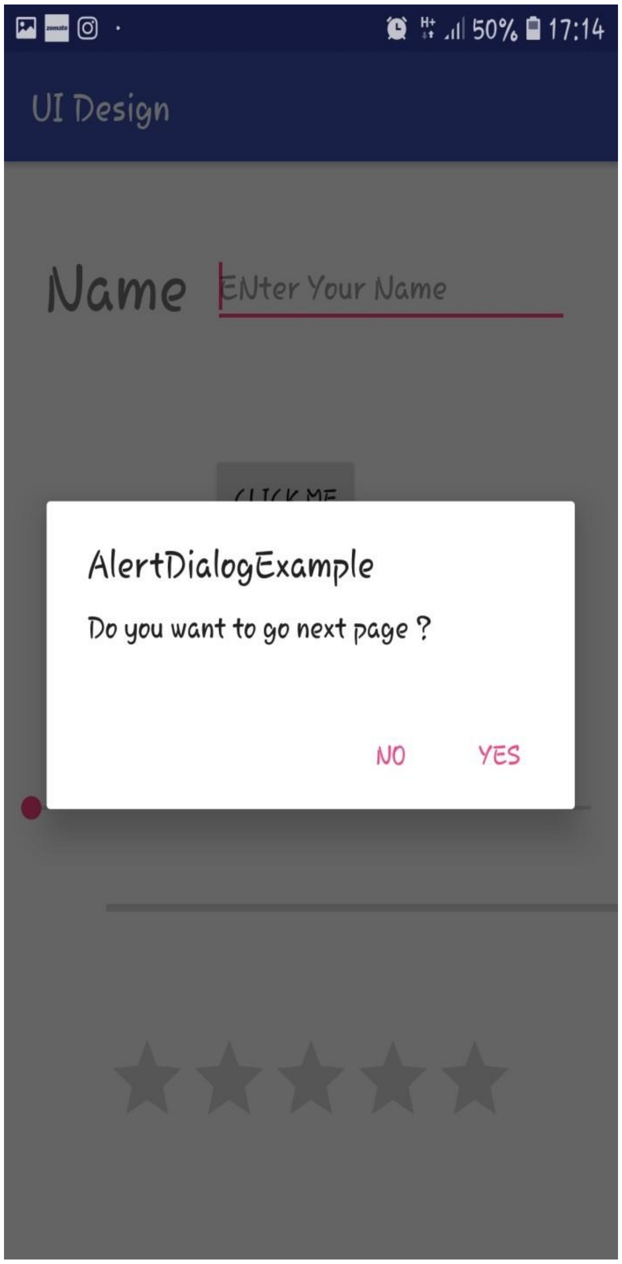
You can use built in vies to Design your activity as per requirement. You just drag and drop the views and set their attributes.
Refer img to design and activity.

Dialogs

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

Dialogs inform users about a task and can contain critical information, require decisions, or involve multiple tasks.


50% 17:13
UI Design
Name
ENter Your Name
CLICK ME
Switch



Menus

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.
Here, we are inflating the menu by calling the **inflate()** method of **MenuInflater** class. To perform event handling on menu items, you need to override **onOptionsItemSelected()** method of Activity class.
There are 3 types of menus in Android:

1. Option Menu:The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a overall impact on the app, such as Search, Compose Email and Settings.
2. Context Menu: A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.
3. Pop-up Menu : A popup menu displays a list of items in a vertical list that is anchored(sticked) to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

How to create a Menu?

For all menu types mentioned above, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource i.e load the XML files as a Menu object in your activity.

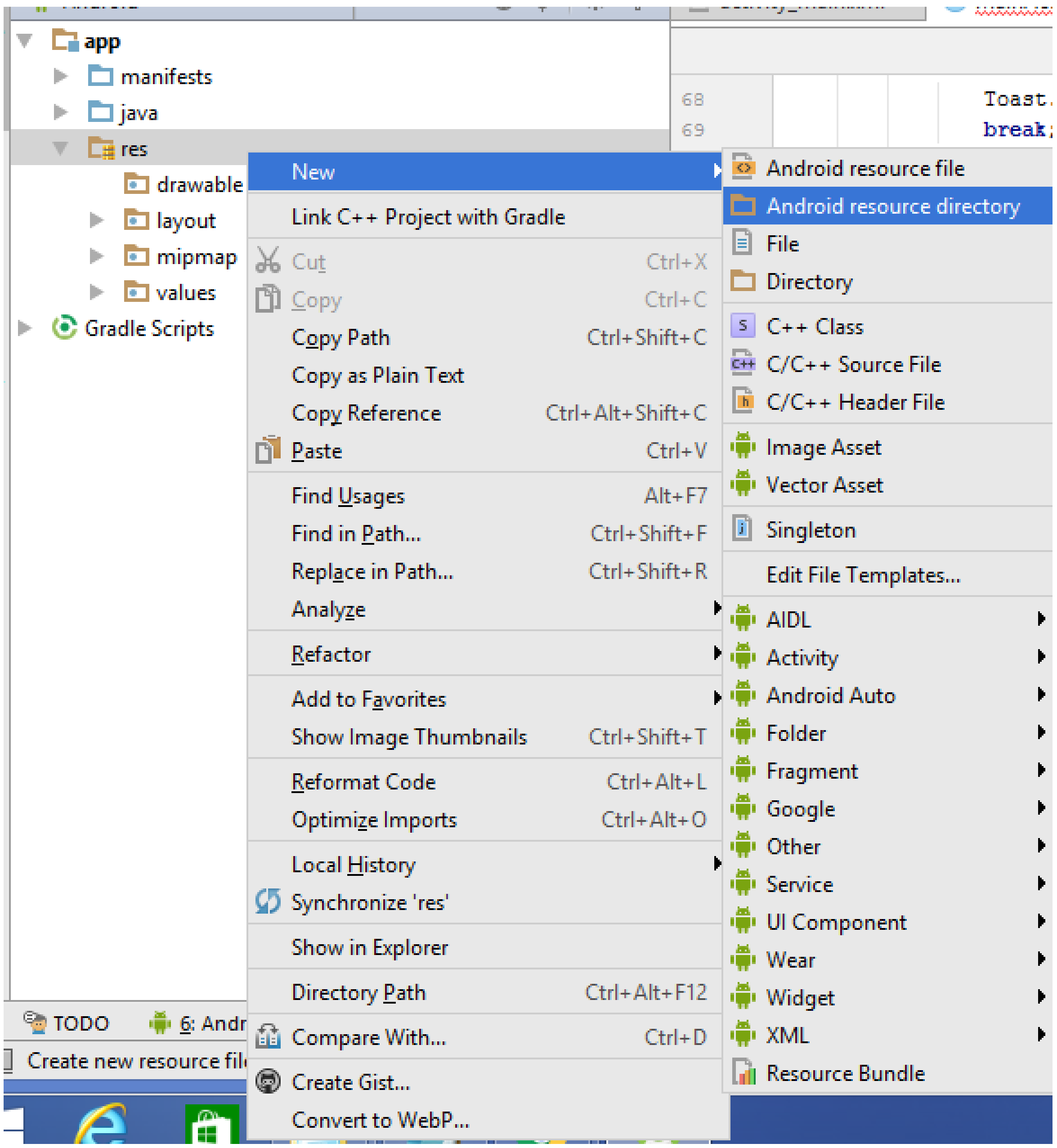
Why to use a separate menu resource?

Using a menu resource is a good practice for a few reasons:

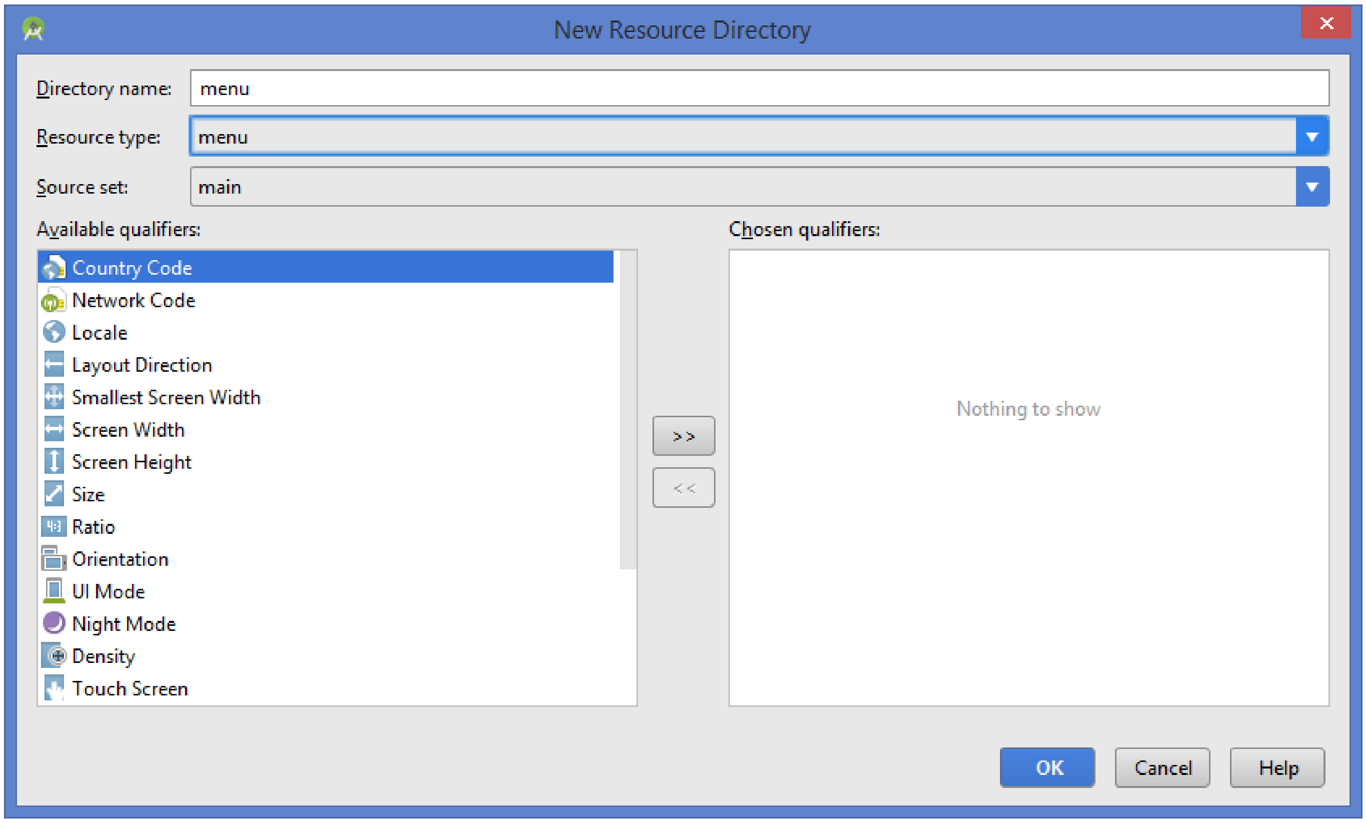
- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

How to create a menu_file.xml file in menu directory?

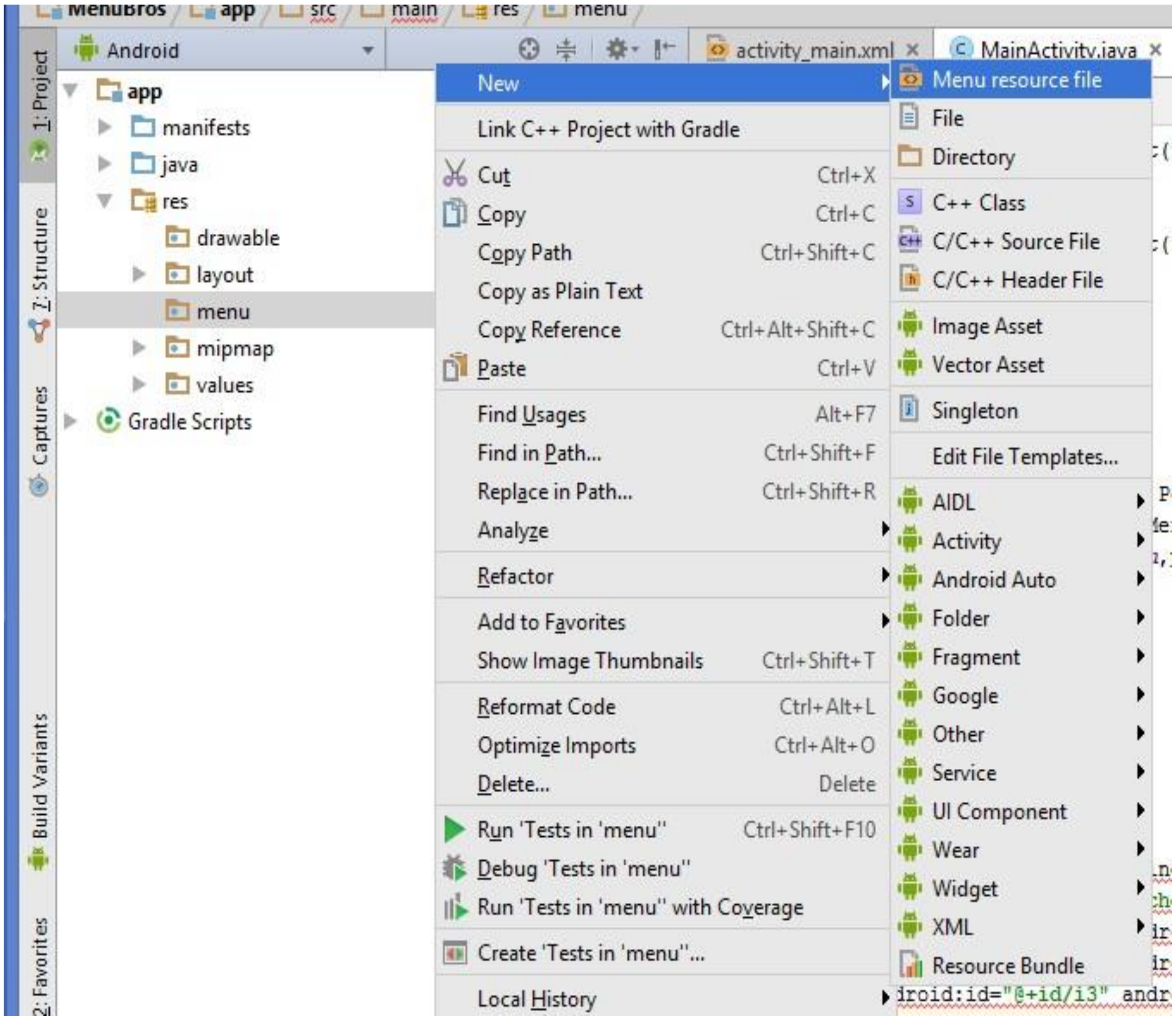
To define the menu_file.xml file, first create a menu directory under res folder. This is done by right clicking on **res --> new --> Android resource directory**.



Then a new window will appear. Type menu in the directory name and choose menu in the Resource type. Then, click on OK.



A new menu directory would be made under res directory. Add menu_file.xml file in menu directory by right clicking on **menu --> New --> Menu resource file**.



Give the name as **menu_file.xml** and click on Ok. The **menu_file.xml** file contains the following tags:

- **<menu>**

  It defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements.

- **<item>**

It creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.

- **<group>**

  It is an optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

menu_file.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

   <item android:id="@+id/i1"
      android:title="item"
       >

      <!-- "item" submenu -->
      <menu>
         <item
            android:title="subitem
      />
         <item
            android:id="@+id/i3"
             />
      </menu>
   </item>
</menu>
```

The <item> element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

- **android:id**

  A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.

- **android:icon**

A reference to a drawable to use as the item's icon.

- **android:title**

   A reference to a string to use as the item's title.

Activity_main2.xml

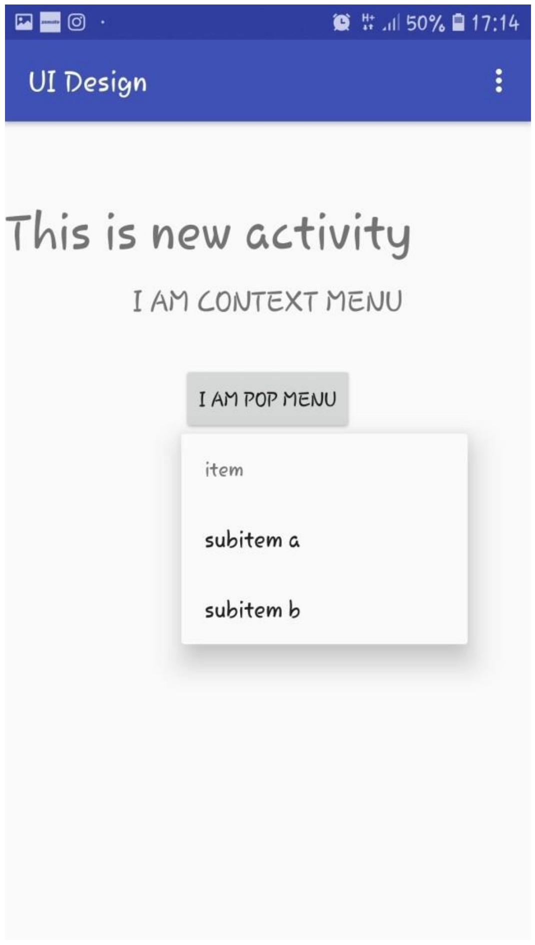
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="comjdjaydeeppatil.trialscoe.Main2Activity">


    <TextView
        android:id="@+id/textView
        "
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="51dp"
        android:textSize="35sp"
        android:text="This is new activity"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        />
    <TextView
        android:id="@+id/t1
        "
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am context menu"
        android:paddingBottom="30dp"
        android:textAllCaps="true"
        android:textSize="20sp"
        android:layout_marginTop="11dp"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        />

    <Button
        android:id="@+id/button
        "
        android:layout_width="wrap_content
        "
```

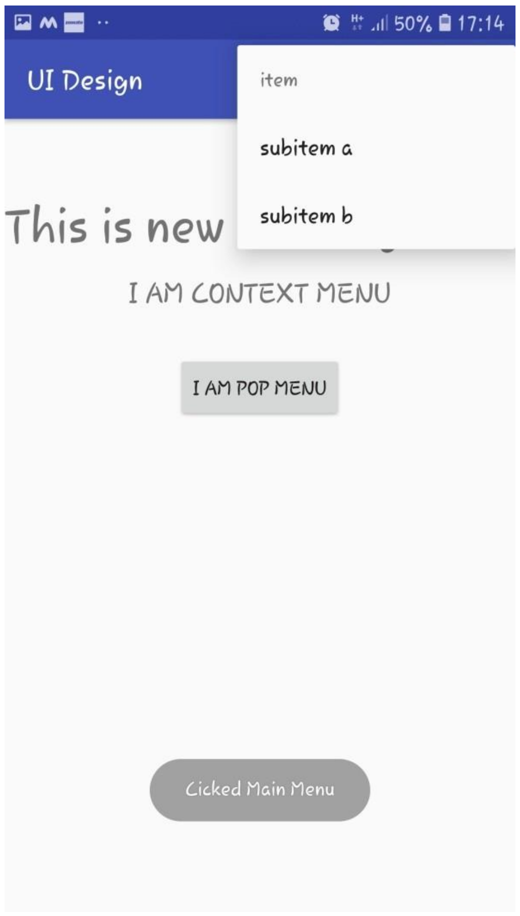
android:layout_height="wrap_conten t" android:onClick="pop" android:text="I am Pop Menu"

```
    android:textAllCaps="true"
    android:layout_below="@+id/textView
    "
    android:layout_centerHorizontal="tru
    e" android:layout_marginTop="68dp"
    />
</RelativeLayout>
```

Main2Activity.java

```
package comjdjaydeeppatil.trialscoe;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.PopupMenu;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class Main2Activity extends

  AppCompatActivity { @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    registerForContextMenu((TextView)findViewById(R.id.t1));

  }


  @Override
  public boolean onCreateOptionsMenu(Menu menu) {

    MenuInflater mi = getMenuInflater();
    mi.inflate(R.menu.menu_file,menu);
    return true;
  }


  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()){
            case R.id.i1:
                Toast.makeText(this,"Cicked Main
                Menu",Toast.LENGTH_SHORT).show(); break;
            case R.id.i2:
                Toast.makeText(this,"I am sub-menu
                1",Toast.LENGTH_SHORT).show(); break;
            case R.id.i3:
                Toast.makeText(this,"I am sub-menu
                2",Toast.LENGTH_SHORT).show(); break;
        }
        return true;
    }


    @Override
    public void onCreateContextMenu(ContextMenu menu, View
v, ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);

        MenuInflater mi = getMenuInflater();
        mi.inflate(R.menu.menu_file,menu);
    }


    @Override
    public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()){
            case R.id.i1:
                Toast.makeText(this,"Clicked Main
                menu",Toast.LENGTH_SHORT).show(); break;
            case R.id.i2:
                Toast.makeText(this,"I am sub-menu
                1",Toast.LENGTH_SHORT).show(); break;
            case R.id.i3:
                Toast.makeText(this,"I am sub-menu
                2",Toast.LENGTH_SHORT).show(); break;
        }
        return true;

    }

    public void pop(View v){
        PopupMenu popup = new PopupMenu(this,v);
        MenuInflater mi = getMenuInflater();
        mi.inflate(R.menu.menu_file,popup.getMenu());
        popup.show();

    }

public boolean onMenuItemClick(MenuItem item)

{
    switch (item.getItemId()){
        case R.id.i1:
            Toast.makeText(this,"Clicked Main
            menu",Toast.LENGTH_SHORT).show(); break;
        case R.id.i2:
            Toast.makeText(this,"I am sub-menu
            1",Toast.LENGTH_SHORT).show(); break;
        case R.id.i3:
            Toast.makeText(this,"I am sub-menu
            2",Toast.LENGTH_SHORT).show(); break;
    }
    return true;


}
}
```





Conclusion:
Thus we studied how to Make Simple UI design using inbuilt views. Also we Studied Menus and Dialog box to make app more attractive.

FAQs:-

1. What's the difference between an implicit and an explicit intent?
2. When should you use a Fragment, rather than an Activity?
3. You're replacing one Fragment with another — how do you ensure that the user can return to the previous Fragment, by pressing the Back button?
4. How would you create a multi-threaded Android app *without* using the Thread class?
5. What is a ThreadPool? And is it more effective than using several separate Threads?
6. What is the relationship between the lifecycle of an AsyncTask and the lifecycle of an Activity? What problems can this result in, and how can these problems be avoided?

Lab. Assignment No – 3

Aim: Android-database Connectivity: Create a SQLite Database for an Android Application and perform CRUD (Create, Read, Update and Delete) database operations.

Objective: To implement stand-alone database (SQLite) as a back end

Theory:

What is SQLite?

SQLite is an SQL Database. So in SQL database, we store data in tables. The tables are the structure of storing data consisting of rows and columns.

What is CRUD?

As the heading tells you here, we are going to learn the CRUD operation in SQLite Database. **But what is CRUD? CRUD** is nothing but an abbreviation for the basic operations that we perform in any database. And the operations are

- **Create**
- **Read**
- **Update**
- **Delete**

Android SQLite

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is "baked into" the Android runtime, so every Android application can create its own SQLite databases.

Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor.

SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema**.

Android SQLite SQLiteOpenHelper

Android has features available to handle changing database schemas, which mostly depend on using the **SQLiteOpenHelper** class.

SQLiteOpenHelper is designed to get rid of two very common problems.

1. When the application runs the first time – At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.
2. When the application is upgraded to a newer schema – Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

SQLiteOpenHelper wraps up these logic to create and upgrade a database as per our specifications. For that we'll need to create a custom subclass of **SQLiteOpenHelper** implementing at least the following three methods.

1. **Constructor**: This takes the Context (e.g., an Activity), the name of the database, an optional cursor factory (we'll discuss this later), and an integer representing the version of the database schema you are using (typically starting from 1 and increment later).

```
public DatabaseHelper(Context context) {

    super(context, DB_NAME, null,

    DB_VERSION);

  }
```

1. **onCreate(SQLiteDatabase db)** : It's called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.
2. **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** : It's called when the schema version we need does not match the schema version of the database, It passes us a **SQLiteDatabase** object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.

We define a DBManager class to perform all database CRUD(Create, Read, Update and Delete) operations.

Opening and Closing Android SQLite Database Connection

Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException {

dbHelper = new DatabaseHelper(context);

database = dbHelper.getWritableDatabase();

return this;

   }
```

The **dbHelper** is an instance of the subclass of

SQLiteOpenHelper. To close a database connection the following

method is invoked.

```
public void close() {

      dbHelper.close();

   }
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc) {

      ContentValues contentValue = new ContentValues();

      contentValue.put(DatabaseHelper.SUBJECT, name);

      contentValue.put(DatabaseHelper.DESC, desc);

      database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);

   }
```

Content Values creates an empty set of values using the given initial size. We’ll discuss the other instance values when we jump into the coding part.

Updating Record in Android SQLite database table

The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc) {

    ContentValues contentValues = new ContentValues();

    contentValues.put(DatabaseHelper.SUBJECT, name);

    contentValues.put(DatabaseHelper.DESC, desc);

    int i = database.update(DatabaseHelper.TABLE_NAME, contentValues, DatabaseHelper._ID
+ " = " + _id,

    null); return

    i;

  }
```

Android SQLite – Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id) {

    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);

  }
```

Android SQLite Cursor

A Cursor represents the entire result set of the query. Once the query is fetched a call to **cursor.moveToFirst()**is made. Calling moveToFirst() does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

The following code is used to fetch all records:

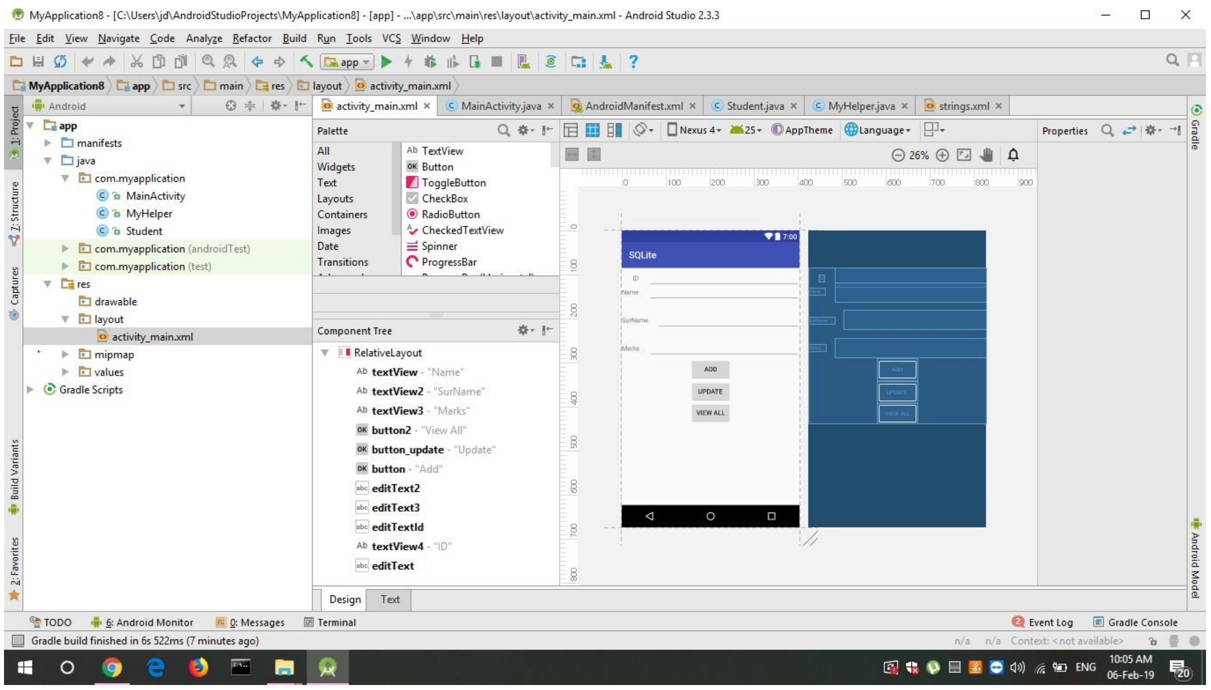
```
public Cursor fetch() {
        String[] columns = new String[] { DatabaseHelper._ID,
                 DatabaseHelper.SUBJECT, DatabaseHelper.DESC };

        Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null, null,
null, null);

        if (cursor != null) {

           cursor.moveToFirst()

           ;

        }

        return cursor;

    }
```

Another way to use a Cursor is to wrap it in a ***CursorAdapter****. Just as* ***ArrayAdapter*** *adapts arrays,* ***CursorAdapter*** *adapts Cursor objects, making their data available to an* ***AdapterView*** *like a* ***ListView****.*

Implementation:

Step 1 – Create new Android project.

Step 2 – Add components in the main activity as shown in the picture below.



activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"

      android:text="Name"
      android:id="@+id/textView"
      android:layout_alignParentTop="true
      "

      android:layout_marginTop="44dp" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"

      android:text="SurName"
      android:id="@+id/textView2"
      android:layout_below="@+id/textView"

      android:layout_marginTop="44dp" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"

      android:text="Marks"
      android:id="@+id/textView3"
      android:layout_below="@+id/textView2"

      android:layout_marginTop="44dp" />


   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="View All"
```

```
    android:id="@+id/button2"
    android:layout_marginTop="46dp"
    android:layout_below="@+id/button"
    android:layout_alignStart="@+id/button" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Update"
    android:id="@+id/button_update"
    android:layout_below="@+id/button"
    android:layout_alignStart="@+id/button" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Add"
    android:id="@+id/button"
    android:layout_marginTop="13dp"
    android:layout_below="@+id/textView3"
    android:layout_centerHorizontal="true"
    />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"

    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_toRightOf="@+id/textView2"
    android:layout_toEndOf="@+id/textView2"
    android:layout_marginLeft="18dp"
    android:layout_marginStart="18dp" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignLeft="@+id/editText"
    android:layout_alignStart="@+id/editText" />
```

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editTextId"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/textView2"
    android:layout_toEndOf="@+id/textView2" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:text="ID"
    android:id="@+id/textView4"
    android:layout_alignBaseline="@+id/editTextId
    "
    android:layout_alignBottom="@+id/editTextId"
    android:layout_alignRight="@+id/textView"
    android:layout_alignEnd="@+id/textView" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/textView4"
    android:layout_alignLeft="@+id/editTextId"
    android:layout_alignStart="@+id/editTextId"></EditText>

</RelativeLayout>
```

Step 3 – Now create a new Java classes called **Student.java** and **MyHelper.java**.

You can see *Project Structure* of above image to know where to add this java classes.

Add the following code in class **Student.java**

<u>Student.java</u>

```
package com.myapplication;

/**
```

```
 * Created by jd on 22-Jan-19.
 */

public class Student {

   private Integer id;
   private String fname;
   private String lname;


   public Student(String fname, String lname)
      { this.fname = fname;
      this.lname = lname;
   }

   public String getFname()
      { return fname;
   }

   public String getLname()
      { return lname;
   }
}
```

Add the following code in class **MyHelper.java**

MyHelper.java

```
package com.myapplication;

/**
 * Created by jaydeep on 27-Sep-17.
 */
import android.content.ContentValues;
import android.content.Context; import
android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class MyHelper extends SQLiteOpenHelper
{
   public static final String DATABASE_NAME= "Student.db"; // DB Name
   public static final String TABLE_NAME = "student_table"; // Table
   Name public static final String COL_1 = "ID"; // Column 1.
   public static final String COl_2 = "Name"; // Column 2.
   public static final String COL_3 = "SurName"; // Column
   3. public static final String COL_4 = "Marks"; // Column
   4.


   public MyHelper(Context context)
   {
      super(context,DATABASE_NAME, null, 1);
      // SQLiteDatabase db = this.getWritableDatabase(); // it will create DB & Table.
   } // So whenever the Constructor will be called , the Database will be created.

   @Override
   public void onCreate(SQLiteDatabase db)
   {
      db.execSQL("Create  Table  "  +  TABLE_NAME   +  "   (ID        INTEGER
                               PRIMARY    KEY AUTOINCREMENT , NAME TEXT,
SURNAME TEXT, MARKS INTEGER) ");
   }

   @Override
   public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1)
   {

   }
   public boolean insertData(String name, String surname, String marks)
   {
      SQLiteDatabase db = this.getWritableDatabase(); // it will create DB & Table.
      ContentValues contentValues = new ContentValues(); // It is used to put the values in the
Column.
      contentValues.put(COl_2,name);
      contentValues.put(COL_3,surname)
      ; contentValues.put(COL_4,marks);
      long result =
      db.insert(TABLE_NAME,null,contentValues); if (result
      == -1)
         return false;
      else
         return true;
   }

   public Cursor getAllData()
   {
      SQLiteDatabase db = this.getWritableDatabase();
      Cursor res = db.rawQuery("select * from " + TABLE_NAME,null);
```

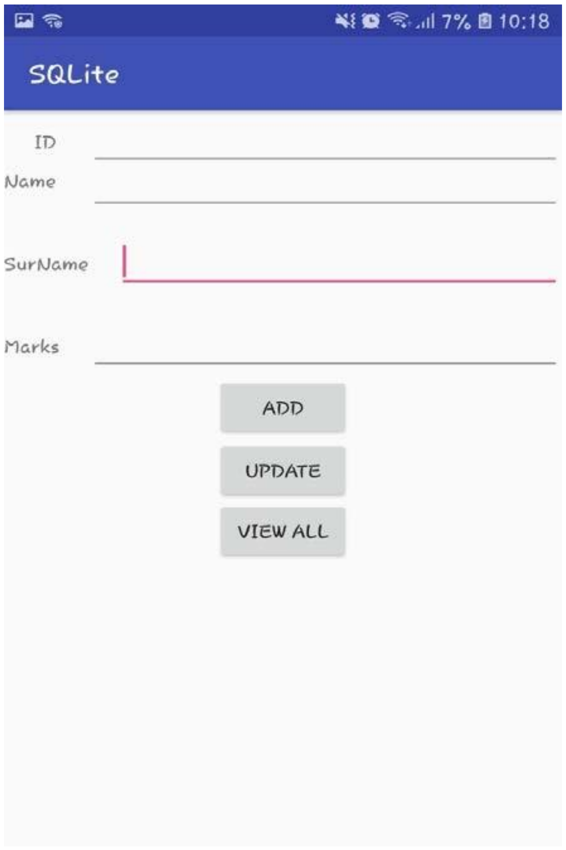
```
        return res;
    }

    public boolean updateData(String id,String name, String surname, String marks)
    {
        SQLiteDatabase db = this.getWritableDatabase(); // it will create DB & Table.
        ContentValues contentValues = new ContentValues(); // It is used to put the values in the
Column.
        contentValues.put(COl_2,name);
        contentValues.put(COL_3,surname);
        contentValues.put(COL_4,marks);
        db.update(TABLE_NAME,contentValues,"ID = ?", new String[]
        {id}); return  true;
    }


}
```

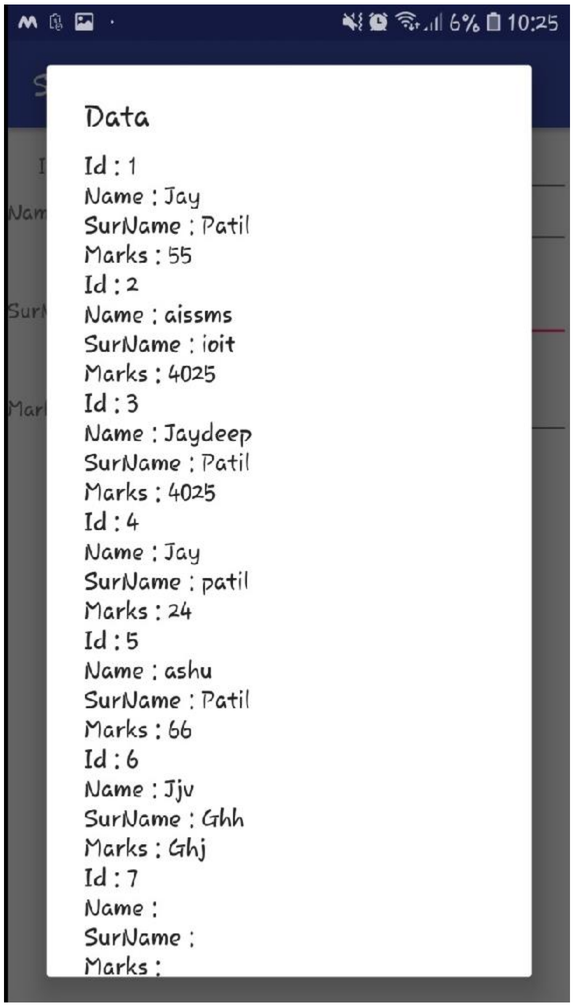
Finally add this code in MainActivity.java file.

MainActivity.java

```
package com.myapplication;


import android.database.Cursor;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
{
    MyHelper mh;
    EditText editName,editSurname, editMarks,editTextId;
    Button btn;
    Button btnViewAll;
    Button btnviewUpdate;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mh = new MyHelper(this); // It is going to call the Constructor of this class.

       editName = (EditText) findViewById(R.id.editText);
       editSurname = (EditText) findViewById(R.id.editText2);
       editMarks = (EditText) findViewById(R.id.editText3);
       editTextId = (EditText) findViewById(R.id.editTextId);
       btn = (Button) findViewById(R.id.button);
       btnViewAll = (Button) findViewById(R.id.button2);
       btnviewUpdate = (Button) findViewById(R.id.button_update);

       addData();
       viewAll();
       updateData();

    }

    public void updateData()
    {

       btnviewUpdate.setOnClickListener(new View.OnClickListener()
          { @Override
          public void onClick(View view)
          {
             boolean isUpdate =
mh.updateData(editTextId.getText().toString(),editName.getText().toString(),editSurname.g
etText().toString(),editMarks.getText().toString());
             if(isUpdate== true)
                Toast.makeText(MainActivity.this,"Data Updated",Toast.LENGTH_LONG).show();

             else
                Toast.makeText(MainActivity.this, " Data Not Updated "
,Toast.LENGTH_LONG).show();
          }
       });
    }

    public void addData()
    {
        btn.setOnClickListener(new View.OnClickListener()
        {

            @Override
            public void onClick(View view) {
                boolean isInserted = mh.insertData(editName.getText().toString()
    ,editSurname.getText().toString(), editMarks.getText().toString());
                if(isInserted == true)
                    Toast.makeText(MainActivity.this,"Data Inserted",Toast.LENGTH_LONG).show();

                else
                    Toast.makeText(MainActivity.this, " Data Not Inserted "
    ,Toast.LENGTH_LONG).show();
            }
        });

      }

      public void viewAll()
      {
        btnViewAll.setOnClickListener(new
            View.OnClickListener() { @Override
            public void onClick(View view)
                {       Cursor       res       =
                mh.getAllData();
                if(res.getCount() == 0)
                {
                    // show msg
                    showMessage("Error","Nothing
                    Found"); return;
                }

                StringBuffer buffer = new
                StringBuffer();
                while(res.moveToNext())
                {
                    buffer.append("Id : " + res.getString(0) + "\n");
                    buffer.append("Name : " + res.getString(1) + "\n");
                    buffer.append("SurName : " + res.getString(2) +
                    "\n"); buffer.append("Marks : " + res.getString(3) +
                    "\n");
                }

                // show all data
                showMessage("Data", buffer.toString());
            }
        });
    }

    public void showMessage(String title , String message)
    {
        AlertDialog.Builder builder = new
        AlertDialog.Builder(this); builder.setCancelable(true);
        builder.setTitle(title);
        builder.setMessage(message);
        builder.show();
    }
}
```

OUTPUT: Screenshots of application





```
Data
Id : 1
Name : Jay
SurName : Patil
Marks : 55
Id : 2
Name : aissms
SurName : ioit
Marks : 4025
Id : 3
Name : Jaydeep
SurName : Patil
Marks : 4025
Id : 4
Name : Jay
SurName : patil
Marks : 24
Id : 5
Name : ashu
SurName : Patil
Marks : 66
Id : 6
Name : Jjv
SurName : Ghh
Marks : Ghj
Id : 7
Name :
SurName :
Marks :
```

Conclusion:

Thus we implement SQLite Application to Add and View and Update records.

FAQs:-

Explain what is SQLite transactions?

List out the areas where SQLite works well?

What is the difference between SQL and

SQLite? Mention what is .dump command is

used for?

Explain how Boolean values in SQLite are

stored? what is the maximum size of a

VARCHAR in SQLite? List out the advantages of

SQLite?

- It does not require separate server processor system to operate
- No setup or administration required SQlite comes with zero-configuration
- An SQLite database can be stored in a single cross-platform disk file
- SQLite is very compact less than 400 KiB
- SQLite is self-contained, which means no external dependencies
- It supports almost all types of O.S
- It is written in ANSI-C and provides easy to use API

Lab Assignment No. 4

Aim:
Design a smart light system which operates controls LED light automatically switched on in evening and gets off in morning using Arduino, LED and LDR interface.

OBJECTIVES:
To design a smart light system which operates controls LED light automatically switched on in evening and gets off in morning by programmable control of a dark resistance and bright resistance of LDR (light dependent resistance) using Arduino IDE and UNO board.

THEORY :

An LDR is a component that has a (variable) resistance that changes with the light intensity that falls upon it. See fig 1 and Fig 2 for LDR view and symbol. This allows them to be used in light sensing circuits.

Fig 1: A typical LDR

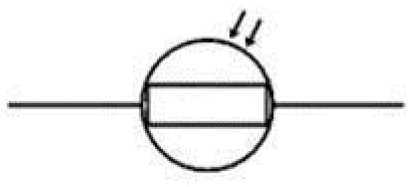
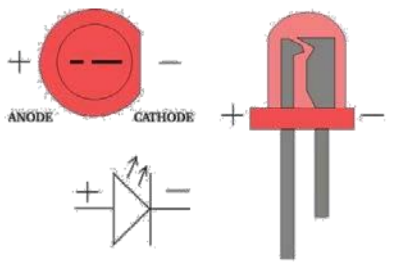


Fig 2 : LDR Circuit Symbol



Negative indicated by flat side of the housing and a short leg ; see fig 3 for and observe LED which you are connecting in this experiment.

Fig 3 : LED Anode and Cathode identification

Variation in resistance with changing light intensity

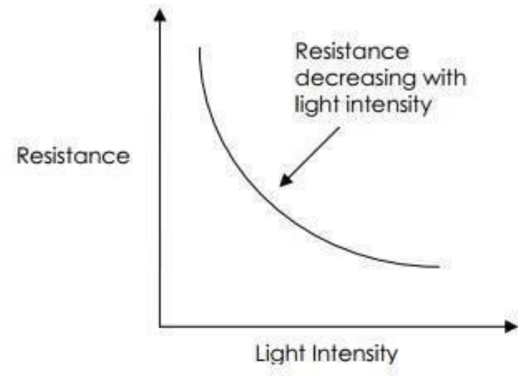


Fig 4 : Typical LDR resistance vs light intensity graph

The most common type of LDR has a resistance that falls with an increase in the light intensity falling upon the device (as shown in the image above). The resistance of an LDR may typically ***for example*** one can observe the following resistances (this also depends on size of LDR and may vary in your case):

Daylight = 5000Ω Dark
= 20000000Ω

You can therefore see that there is a large variation between these figures. If you plotted this variation on a graph you would get something similar to that shown by the graph shown above.

Light dependent resistance (LDR) values shows different resistance values in dark and bright light see Fig 4 which indicate same thing using graph. This can be observed just by measuring LDR resistance on multimeter by exposing LDR to bright light and recording the value of resistance and other case by holding it in dark. These two thresholds are deciding dark and bright resistance. So if we want to put on LED in evening then on port where LED is connected need to put ON if dark resistance value or greater than value is available on analog port where LDR is connected. In else part or by excessively specifying bright resistance value one can put off LED by wring vice vers statement for LED light port. One cam also observes the values on serial port by appropriate statements. The connections are shown in fig 5.

1. First connect LDR to any analog port (out of any six) of Arduino UNO board.
2. Connect LED to any digital pin (out of any eleven) of Arduino UNO board.
3. Observe values on serial port
4. Write statements in of Arduino IDE as per specifying dark and bright resitance value to put ON/OFF LED.
5. Hold LDR in palm and cover it by all fingers; this time LED need to glow. Now expose LDR to light and see; LED need to turn off automatically.
6. By looking room light bright resistance; by turning ON and OFF of bulb / Tube light one can see same effect on automatic LDE ON /OFF.

INPUT : LED (1 qty.)

LDR (1 qty.)
One Arduino UNO
Board Wires for
connection

OUTPUT : On Serial port of Arduino IDE for dark and bright resistance value.
Actual LED ON / OFF based on specified dark and bright resistance values .

Connection

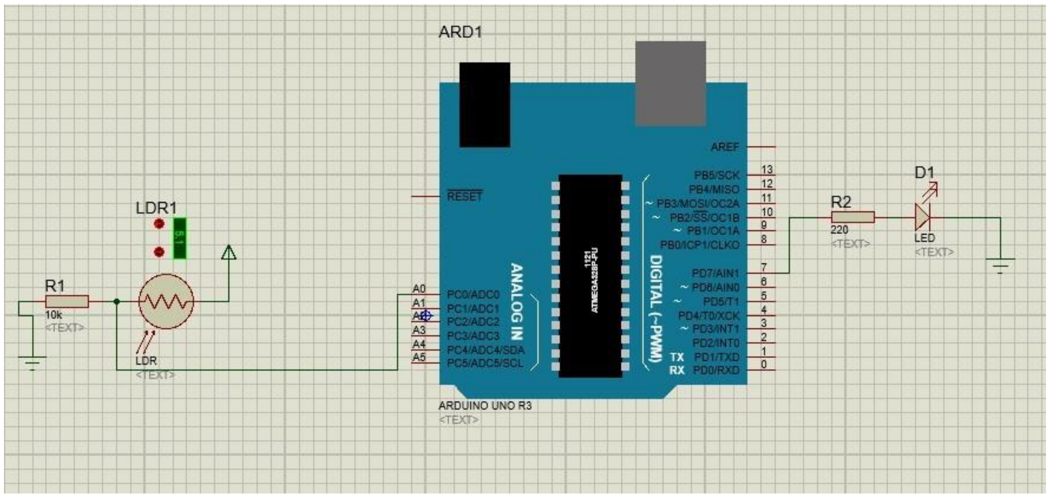


Fig 5: Interfacing LDR and LED on Arduino board

CONCLUSION : This way we make can use the open source platforms like Arduino Uno and its IDE to make build smart lighting system based using properties of LDR sensor that controls LED ON/OFF operation automatically by turning it ON in dark and turning it OFF in bright light.

FAQ :

1. What is LDR?
2. How LDR functions?
3. What is NTC and PTC and what is its significance in real world? Is it possible to manage this from software to do vice versa operation of LDR?
4. Is it possible to control different 2 different LEDs connected to different digital pins of Arduino on the basis of different dark and bright conditions of LDR? If ‘yes’ how? If ‘no’ why?

Lab Assignment No. 5

Aim:
Design and Develop a GUI for FAN regulator that uses Android platform.

Objective:
To develop a GUI for fan regulator which can be handled by android application.

Theory:

What is Android GUI:

User interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

Implementation:

MainActivity.java

```
package com.example.fan;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.RotateAnimation;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    ImageView imageView ;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imageView = (ImageView)findViewById(R.id.fanImage);

    }
    public void changeSpeed(View view) {
        Animation animation =
 AnimationUtils.loadAnimation(getApplicationContext(), R.anim.rotate);
        System.out.println(view.getId());
        switch (view.getId()) {

            case (R.id.speed0):
                System.out.println(R.id.speed0);
                animation.setDuration(Integer.MAX_VALUE);
                break;
            case (R.id.speed1):
                animation.setDuration(150);
                break;
            case (R.id.speed2):
                animation.setDuration(100);
                break;
            case (R.id.speed3):
                animation.setDuration(50);
                break;
        }
        animation.setRepeatCount(Animation.INFINITE);
        imageView.setAnimation(animation);
        imageView.startAnimation(animation);
    }
}
```

Acivity_main.xml

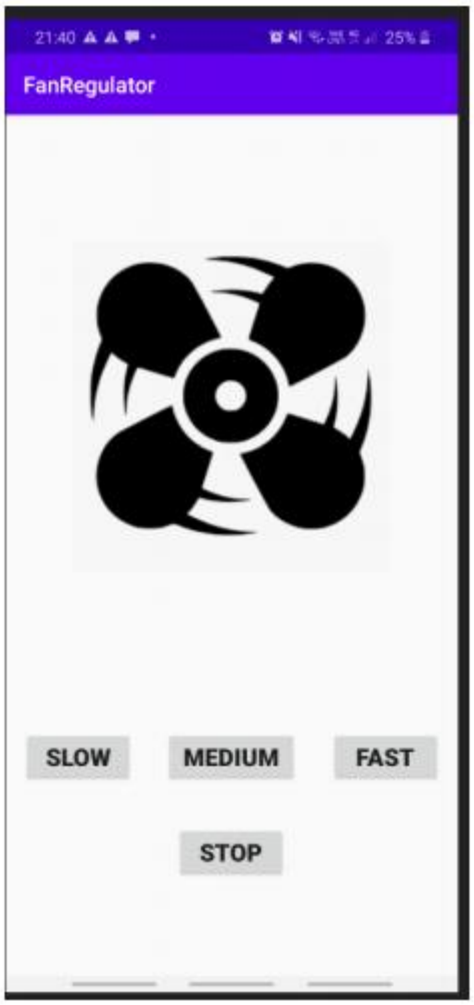
```
    <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">


            <ImageView
                android:layout_width="300dp"
                android:layout_height="300dp"
                android:id="@+id/fanImage"
                android:src="@drawable/fan1"
                android:layout_marginLeft="50dp"
                android:layout_marginRight="50dp"
                android:layout_marginTop="150dp"
                android:layout_marginBottom="150dp"
                />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <Button
                    android:id="@+id/speed0"
                    android:layout_width="50dp"
                    android:layout_height="wrap_content"
                    android:layout_marginLeft="50dp"
                    android:onClick="changeSpeed"
                    android:text="0" />

                <Button
                    android:id="@+id/speed1"
                    android:layout_width="50dp"
                    android:layout_height="wrap_content"
                    android:layout_marginLeft="30dp"
                    android:onClick="changeSpeed"
                    android:text="1" />
```

```
                <Button
                    android:id="@+id/speed2"
                    android:layout_width="50dp"
                    android:layout_height="wrap_content"
                    android:layout_marginLeft="30dp"
                android:onClick="changeSpeed"
                android:text="2" />

            <Button
                android:id="@+id/speed3"
                android:layout_width="50dp"
                android:layout_height="wrap_content"
                android:layout_marginLeft="30dp"
                android:onClick="changeSpeed"
                android:text="3" />

        </LinearLayout>


</RelativeLayout>
```

Rotate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <rotate
        android:toDegrees="75"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="200"
        android:repeatMode="restart"
        android:repeatCount="infinite"

        />

</set>
```

OUTPUT



Conclusion: **A simple android animation for FAN Regulator has been created.**

Lab Assignment No. 6

Aim:
Design an Android based FAN regulator which operates controls Dynamic Fan Control using NodeMUC ESP8266(wireless transceiver).

OBJECTIVES :
To design an Android based FAN regulator which operates controls Dynamic Fan Control automatically using Android APP(BLYNK) by programmable control in Widget Box selecting Slider using Arduino IDE and NodeMUC board.

THEORY :
A motor is an electrical machine which converts electrical energy into mechanical energy. The **principle of working of a DC motor** is that "*whenever a current carrying conductor is placed in a magnetic field, it experiences a mechanical force"*. The direction of this force is given by Fleming's left hand rule and its magnitude is given by F = BIL. Where, B = magnetic flux density, I = current and L = length of the conductor within the magnetic field.

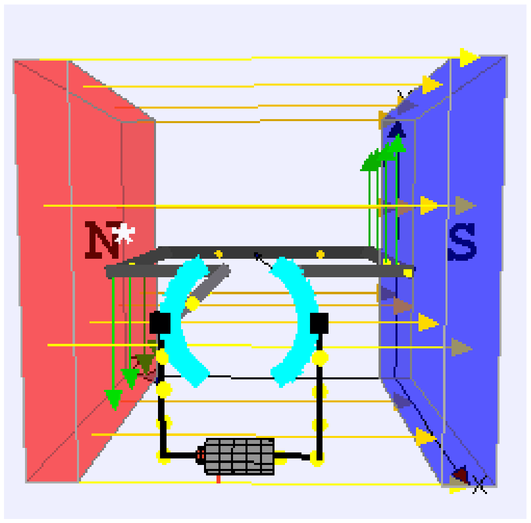


Fig 1 : Working of DC Motor

Fleming's left hand rule: If we stretch the first finger, second finger and thumb of our left hand to be perpendicular to each other AND direction of magnetic field is represented by the first finger, direction of the current is represented by second finger then the thumb represents the direction of the force experienced by the current carrying conductor.

Above animation helps in understanding the

Working principle of a DC motor. When armature windings are connected to a DC supply, current sets up in the winding. Magnetic field may be provided by field winding (electromagnetism) or by using permanent magnets. In this case, current carrying armature conductors experience force due to the magnetic field, according to the principle stated above. Commutator is made segmented to achieve unidirectional torque. Otherwise, the direction of force would have reversed every time when the direction of movement of conductor is reversed the magnetic field.

INPUT : DC MOTOR(1 qty.)
One NodeMUC Board
Wires for connection

OUTPUT : Actual FAN speed is is control by adjusting the slider in BLYNK app.

Connection : As shown in fig 2 the Interfacing DC motor on NodeMUC . Blynk App screen are shown in Fig 3 ; which is used to control FAN or any output connected to NodeMUC remotely .

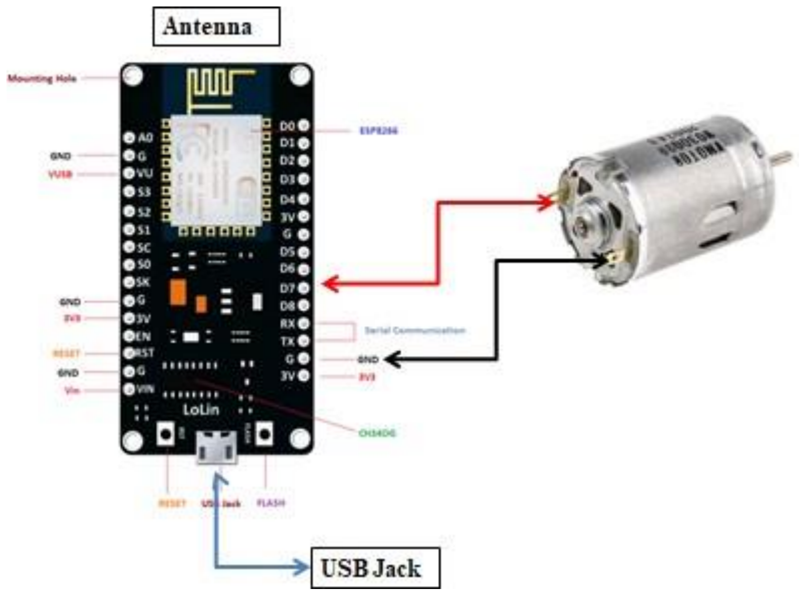


Fig 2 : Interfacing DC motor on NodeMUC board

/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include
<BlynkSimpleEsp8266.h>
// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "24b895ca4e4645d29f0f19b3047b33c6";
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = “Navjyot";

```
char pass[] = “waheguru";
void setup()
{
  // Debug console
  Serial.begin(9600);
  Blynk.begin(auth, ssid,
  pass);
  // You can also specify server:
  //Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 80);
  //Blynk.begin(auth, ssid, pass, IPAddress(192.168.1.100), 8080);
}
void loop()
{
  Blynk.run();
}
```

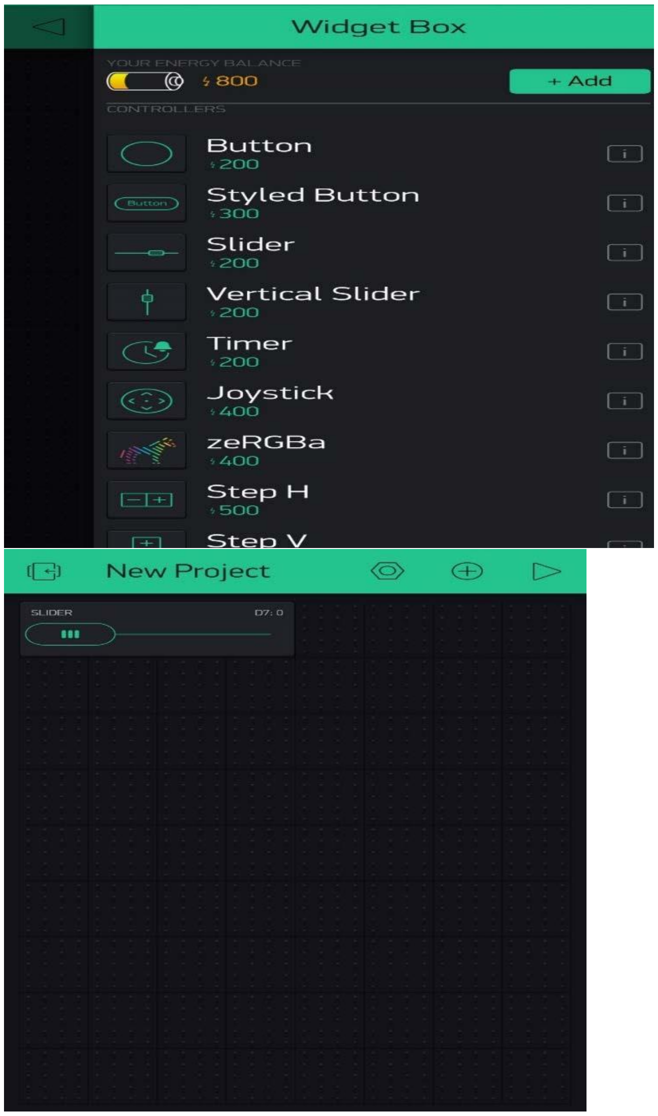


Fig 3 : Blynk App GUI

CONCLUSION : In this experiment we verified by interfacing DC motor on NodeMUC and it is observed that DC motor can be controlled from wireless domain with authentic IP address.

FAQ :

1] Differentiate NodeMUC Vs Arduino UNO.
2] What is purpose of Rx and Tx pin in NodeMUC?
3] Is it possible to control 230V; 50Hz operated Fan? 4] How relay operate?

Lab Assignment No. 7

Aim:

Wireless Network: Develop an app for a rolling display program of news on computer display. The input strings are supplied by the mobile phone/ by another computer connected through wireless networks.

Objective:

To design a wireless network for transferring data to a rolling display from computer.

Theory:

Wireless LED Moving Display Board:

•Wireless Scrolling Message Electronic Display Board is an embedded solution, by which one can display the messages, received as SMS. Such displays can be kept in different parts of the city. The public can send flash information from Mobiles or PCs to these displays for instant viewing.

•Wireless Display Boards are designed with LED's to quickly display large amounts of information in different bright colors - red, green and amber, resulting in great visual impact, superior visibility, and high resolution. Any messages sent from a PC or mobiles will be displayed in all the display Boards installed in any part of cities or country.

Benefits:

- No More Same Advertisement again and again for days/months
- Instant, Current and Hot Topics reach the Public immediately
- Communicate critical information to employees immediately
- Control multiple Displays from a centralized location
- Dynamic Advertisement for Recurring Revenue
- Common Display system which displays instant messages like Flash News, in Places like exhibitions, Roadside Hoardings
- Instant Message Delivery from Head offices to branches or field stations like Petrol Bunks
- Easy to change messages remotely compared to commonly available Moving Display systems where it is difficult to access

Application areas:

- Advertisement Hoardings
- At Railway Stations/Bus stops for displaying train/bus schedule
- Instant update of Petrol Prices to all petrol Bunks from a Central office
- Stock Tickers, which displays dynamically current value of the Stocks
- Different Stalls, Multiplexes, Malls & Stores for displaying product ranges
- Public information on local society
- Current Prices of Commodities at different parts of the country

- At Religious Places like Ashrams or Temples where common messages will have to be displayed at all Branches spread across the country

Other details:

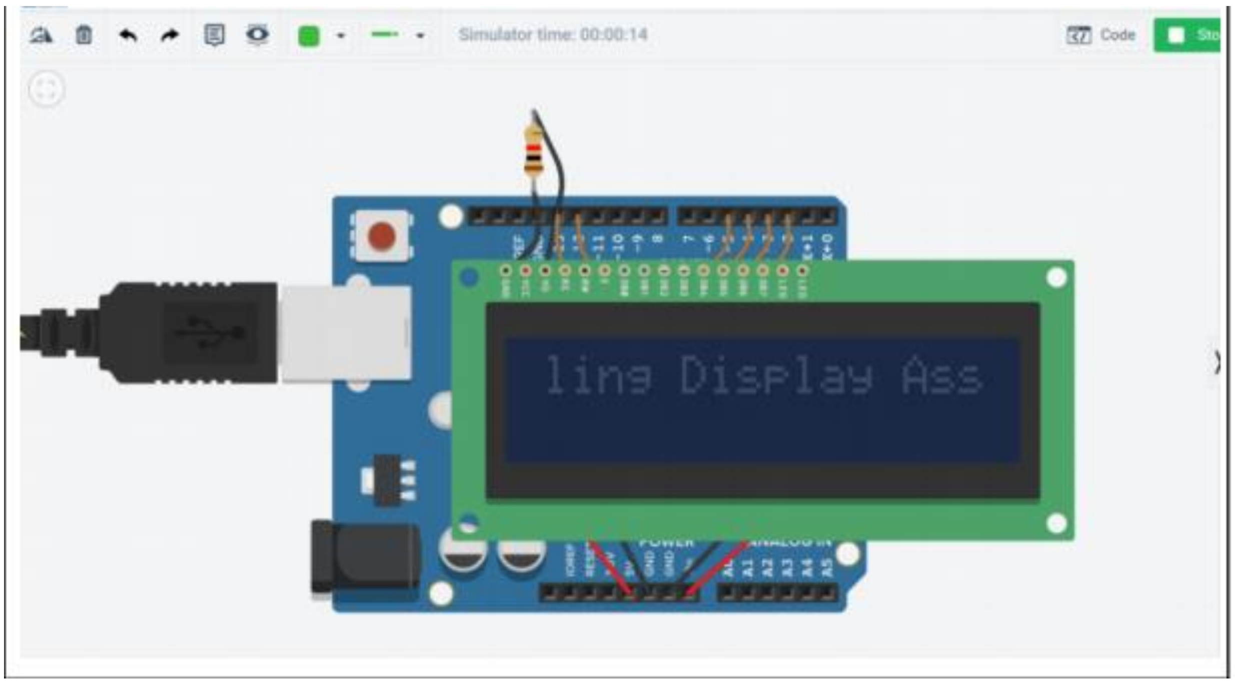
- Scrolling or Fixed Messages
- Programmable Power On Default Message
- Non Volatile messages stored capacity
- Messages only From Authorized Mobile numbers
- Display Messages as per schedule
- Configurable through PC
- Configurable through SMS
- Programmable using SMS from Master
- Single or Multi line Display

Power Consumption	230 v 5 A
Dimension	24.6" H x 5.1" W x 1.5" D,(625 H x 130 W x 38 mm D)
Letter Size	4 inch
Optimum Viewing Angle	30 meter
Input Voltage	220-240 V AC
Operating Temperature	32 Degree F to 158 Degree F (0 Degree C to 70 Degree C)
Display Technology	Dot Matrix
Power Source	AC
Character Capacity	1000
Pixel Density	10 mm
Country of Origin	Made in India
Display	16X2 LCD Display
Power Supply	12 VDC/3A
Operating Humidity	5 to 9 percent relative humidity non-condensing
Minimum Order Quantity	1 Number

Implementation:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(13,12,5,4,3,2);
void setup() {
lcd.begin(16, 2);
lcd.print("Rolling Display Assignment !");
delay(1000);
}
void loop() {
for (int i = 0; i < 28; i++) {
lcd.scrollDisplayLeft();
delay(150);
}

for (int i = 0; i <44; i++) {
lcd.scrollDisplayRight();
delay(150);
}

for (int i= 0; i< 28; i++) {
lcd.scrollDisplayLeft();
delay(150);
}
delay(2000);
}
```

Output:



Conclusion:

A wireless display has been designed to display messages.

Lab Assignment No. 8

AIM:

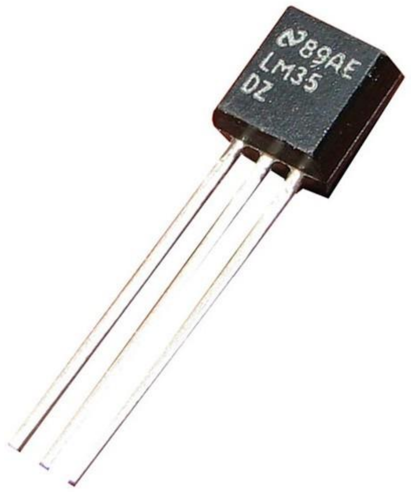
Design a sophisticated system, which acquires data from multiple sensors and transmits it through a wireless module via Arduino, and will be received by another Arduino and given to the PC for analysis using Machine learning.

OBJECTIVES:

To Acquire temperature values from 4 LM35 sensors and 1 DHT11 temperature+humidity sensor in real time and transmit that data to the remote Arduino wirelessly using NRF24l01+ transceiver modules and further give that data to PC via serial communication and analyze it using Machine Learning in Python.

THEORY:

A] <u>LM35 Sensor</u>:



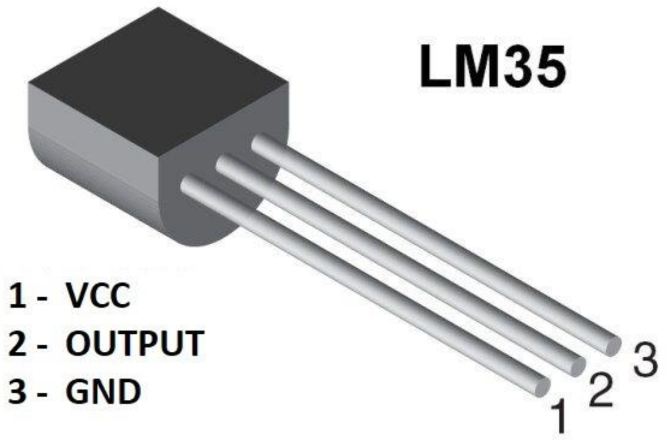


Fig 1.1 LM35 sensor   Fig 1.2 LM35 Pinout

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature. The LM35 device has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling. The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C, over a full −55°C to 150°C temperature range. Lower cost is assured by trimming and calibration at the wafer level. The low-output impedance, linear output, and precise inherent calibration of the LM35 device makes interfacing to readout or control circuitry especially easy. The device is used with single power supplies, or with plus and minus supplies. As the LM35 device draws only 60 µA from the supply, it has very low self- heating of less than 0.1°C in still air. The LM35 device is rated to operate over a −55°C to 150°C temperature range, while the LM35C device is rated for a −40°C to 110°C range (−10° with improved accuracy).

Features of LM35 are as follows:

- Calibrated Directly in Celsius (Centigrade)
- Linear + 10-mV/°C Scale Factor

- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full −55°C to 150°C Range
- Suitable for Remote Applications
- Low-Cost Due to Wafer-Level Trimming
- Operates From 4 V to 30 V
- Less Than 60-μA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Non-Linearity Only ±¼°C Typical
- Low-Impedance Output, 0.1 Ω for 1-mA Load

Applications:

- Power Supplies
- Battery Management
- HVAC
- Appliances

For detail description please refer the datasheet - http://www.ti.com/lit/ds/symlink/lm35.pdf

B]<u>DHT11 Sensor</u>:

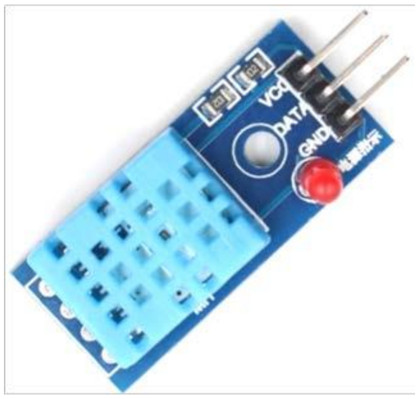


Fig. 2.1 DHT11

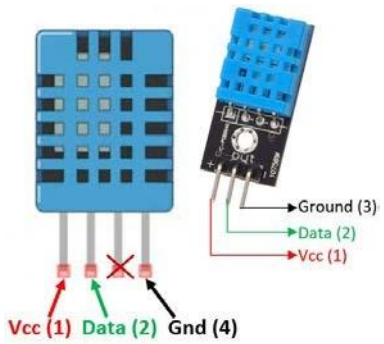


Fig 2.2 DHT11 pinout

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost- effectiveness.
Features of DHT11 are as follows:

- Operating Voltage: 3.5V to 5.5V

- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

Applications:

- Measure temperature and humidity
- Local Weather station
- Automatic climate control
- Environment monitoring

For detail description please refer the datasheet - https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

C]<u>nRF24L01+ wireless transceiver:</u>

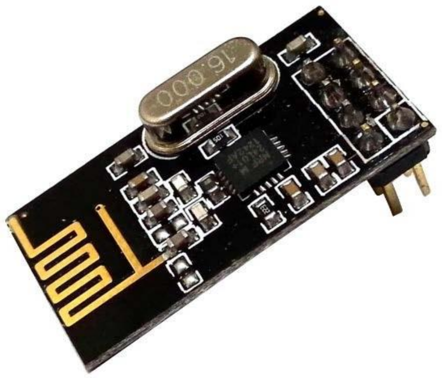


Fig 3.1 nRF24L01+

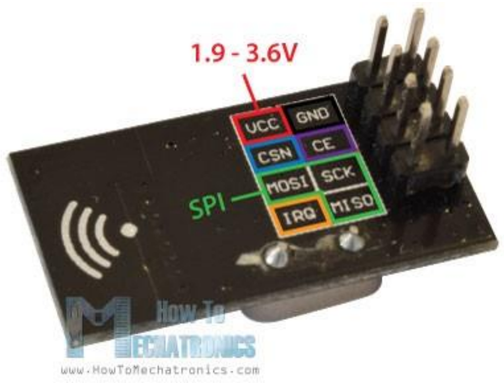


Fig 3.2 nRF24L01+ pinout

The nRF24L01+ is a single chip 2.4GHz transceiver with an embedded baseband protocol engine, suitable for ultra-low power wireless applications. The nRF24L01+ is designed for operations in the world-wide ISM frequency band at 2.4-2.4835GHz. To design a radio system with the nRF24L01+, you simply need an MCU and a few external passive components. One can operate and configure the NRF24L01+ through a Serial Peripheral Interface (SPI). The register map, which is accessible through the SPI, contains all configuration registers in the nRF24L01+ and is accessible in all operation modes of the chip.

Features of nRF24L01+ are as follows:

- World Wide 2.4GHz ISM band Operation
- 1 to 32 bytes dynamic payload size per packet
- Integrated voltage regulator
- 1.9V to 3.6V Supply Range
- 250kbps, 1 and 2 Mbps air data rate

For detail description please refer the datasheet - https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

D] Machine Learning:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

Support Vector Machine:

A Support Vector Machine models the situation by creating a *feature space*, which is a finite-dimensional vector space, each dimension of which represents a "feature" of a particular object. In the context of spam or document classification, each "feature" is the prevalence or importance of a particular word. The goal of the SVM is to train a model that assigns new unseen objects into a particular category. It achieves this by creating a linear partition of the feature space into two categories. Based on the features in the new unseen objects (e.g. documents/emails), it places an object "above" or "below" the separation plane, leading to a categorisation (e.g. spam or non-spam). This makes it an example of a non-probabilistic linear classifier. It is non-probabilistic, because the features in the new objects fully determine its location in feature space and there is no stochastic element involved.

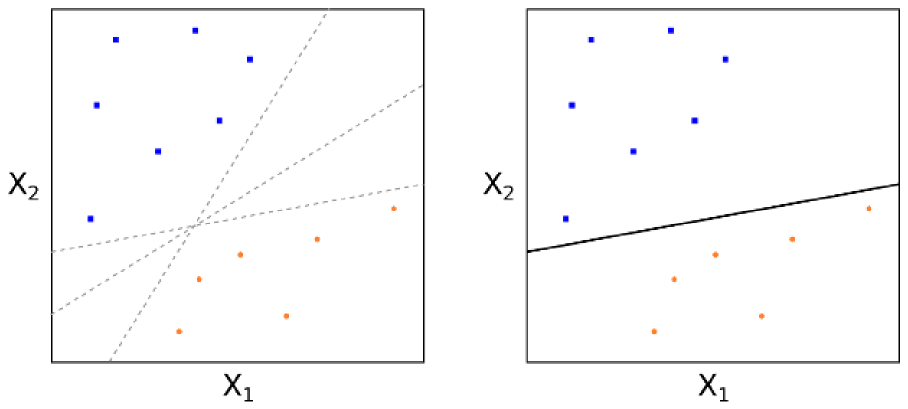


Fig 4 Example of Binary Support Vector Classifier

Procedure:

1. Transmitter - Connect the 4 LM35 sensors to the A0-A3 pins of the Arduino. The DHT11 to the D7 pin and the Nrf24L01+ to the respective pins from D9 to D13 as given in the official library.
2. Receiver – Connect the nRF24L01+ exactly same as that of the transmitter.
3. Download the libraries for dht11 and nRF24L01+ from the 'manage libraries' of the Arduino IDE.
4. Upload the respective code into the transmitter and receiver Arduino Boards s and check the output on the serial monitor if the Arduino IDE.
5. The same output is given to the machine learning algorithm in Python which has been have trained to classify the atmosphere in real time.
6. We use a simple Support Vector Classifier, which is the widely used algorithm in machine learning. It can classify the atmosphere with a high accuracy.

Connection with Arduino UNO Board:

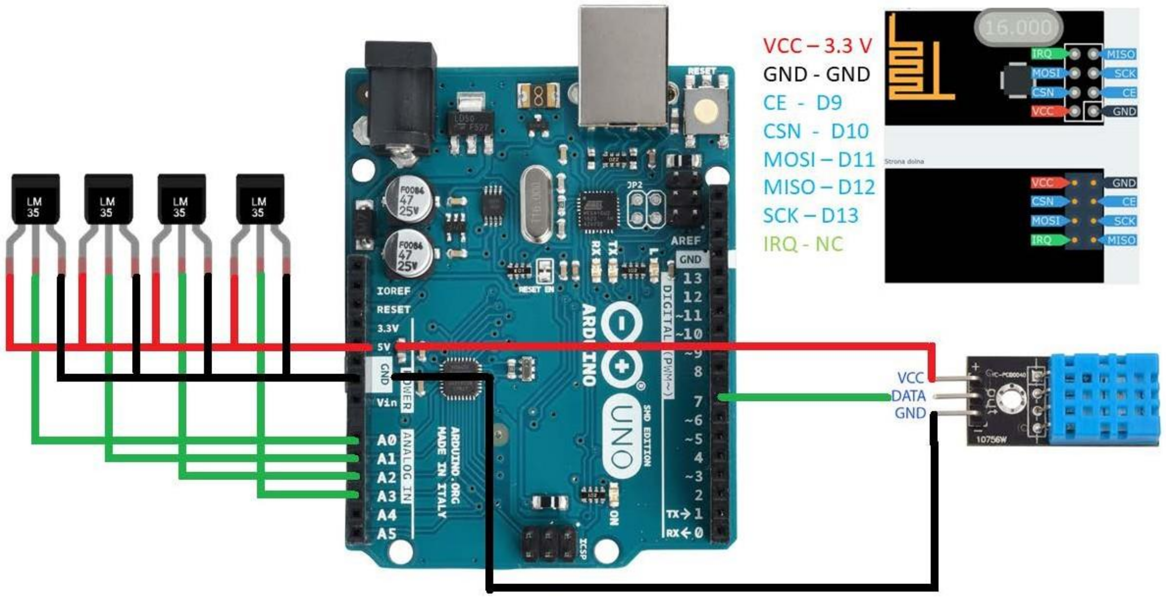


Fig 5 Transmitter Connections

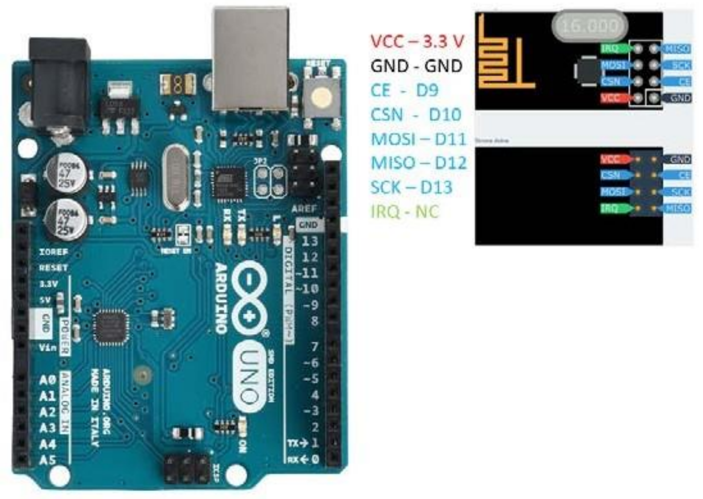
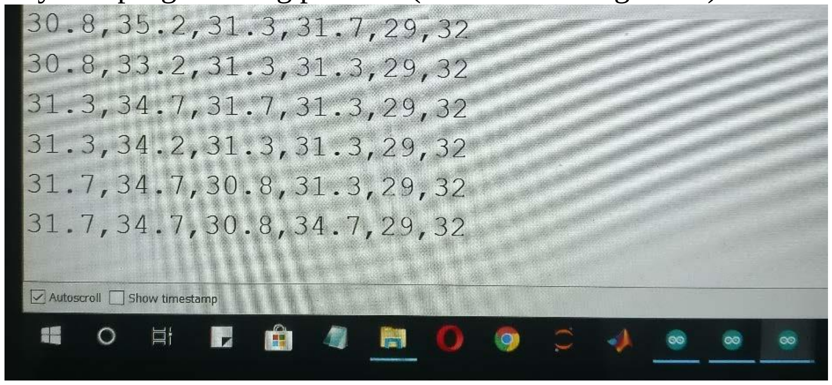


Fig 6 Receiver Connections

INPUT : LM35 analog temperature sensors – 4 Nos.
DHT11 module – 1
Nos. NRF24L01+ -- 2
Nos.
Arduino UNO Boards – 2 Nos

OUTPUT : Arduino IDE Serial Monitor
Python programming platform(Prediction using SVM)



```
30.8,35.2,31.3,31.7,29,32
30.8,33.2,31.3,31.3,29,32
31.3,34.7,31.7,31.3,29,32
31.3,34.2,31.3,31.3,29,32
31.7,34.7,30.8,31.3,29,32
31.7,34.7,30.8,34.7,29,32
```

Fig 7 Serial Monitor Arduino - Comma separated 5 consecutive Temperature Values in *Celcius and 1 Humidity Value in Percentage

CONCLUSION :
This way we make can use of the open source platforms like Arduino and Python to make our own data and use for machine learning applications in real time without any limitations. The sensors can be varied and different machine learning training algorithms can be implemented for higher accuracy and performance.

FAQ :

1. 1What is the drawback of LM35 sensor?
2. 2What is the limitation of DHT11 sensor?
3. 3Why use nRF24L01+, when we have Bluetooth and XBEE modules?

What are the advantages and disadvantages?

4. What is machine learning? Why do we use it solve real world problems? Compare with classical approach.