**Grow with Google Challenge Scholarship**

**Lesson 2 Notes**

**Are you ready?**

This course assumes you are comfortable working in Java. In addition we assume that you have some basic familiarity with XML (or HTML), [**GitHub**](https://github.com/), networking and threading.

**What do I do if I don't have these prerequisites?**

Don't fret! We have other Android and Java related courses.

* [**Android Development for Beginners**](https://www.udacity.com/course/android-development-for-beginners--ud837): This is the first course in a series of free courses about Android development. It assumes no programming knowledge and is perfect for the beginner student who wants to jump quickly into making an Android app. You'll learn Android in parallel to learning basic Java. After this course, you may wish to take our other beginner courses for [**User Input**](https://www.udacity.com/course/android-basics-user-input--ud836), [**Multi-Screen Apps**](https://www.udacity.com/course/android-basics-multi-screen-apps--ud839), [**Networking**](https://www.udacity.com/course/android-basics-networking--ud843) and [**Data Storage**](https://www.udacity.com/course/android-basics-data-storage--ud845).
* [**Android Basics Nanodegree Program**](https://www.udacity.com/course/android-basics-nanodegree-by-google--nd803): To add features and support on top of our free courses, sign up for the Android Basics Nanodegree Program to get access to ten projects, code reviews, personalized feedback, one-on-one appointments and moderated forums. If you'd like to build a portfolio of Android projects and have the support of Udacity mentors and staff, this option is for you.
* [**Java Programming Basics**](https://www.udacity.com/course/java-programming-basics--ud282): If you'd like an introduction to pure Java programming, consider taking this course, a deep dive into the basics of Java.
* [**Git and Github**](https://www.udacity.com/course/how-to-use-git-and-github--ud775): If you're not sure what GitHub is all about, take this course to help you manage your code and collaborate with other developers.
* [**Android Development for Beginners - Lesson 1**](https://www.udacity.com/course/android-development-for-beginners--ud837): If you've never worked with a language like XML or HTML but you're familiar with Java, consider taking the first lesson of the Android Development for Beginners class. It is solely focused on XML layouts for Android.

# A Brief Introduction to Android Studio

In this course we strongly recommend using [**Android Studio**](https://developer.android.com/studio/index.html) as your development environment. It is the defacto IDE for Android Development, and the environment we use in all videos in the course.

Android Studio is specifically made for creating Android Apps. It is actively supported and maintained by Google. It even contains an Android emulator so that you can start creating Android Apps without needing a real device. Jamal Eason will describe a few more features which will be helpful on your journey to becoming an Android Developer.

Check out developer.android.com for a [**full feature list for Android studio**](https://developer.android.com/studio/features.html).

## Installing Android Studio

You should install Android Studio now, and set it up to use either a physical Android device or an emulated phone. If you need guidance for installing Android Studio, there is a mini course with instructions for both Windows and Mac installations which will also go through how to setup an emulator and make your first Hello World app.

## [Go to the Android Studio Installation Mini Course](https://classroom.udacity.com/courses/ud808/lessons/4216368924/concepts/43072785890923)

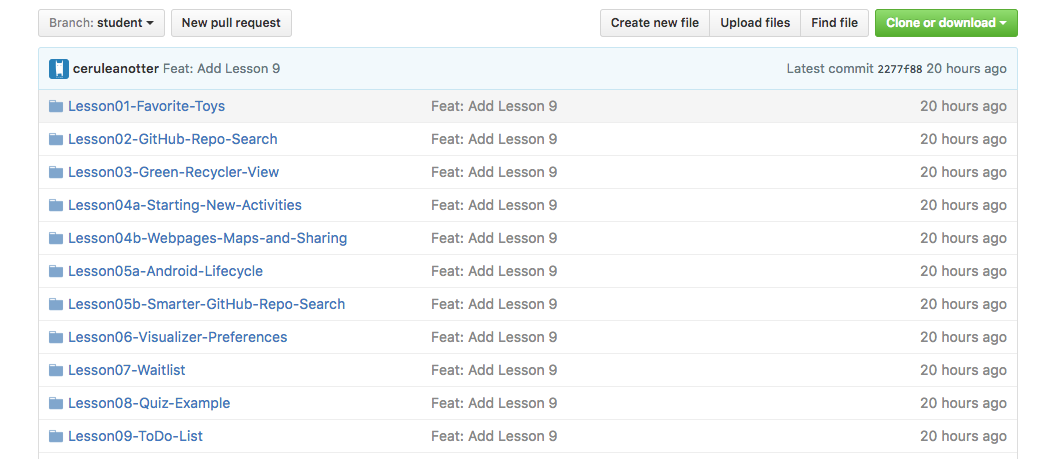
# Code Repositories

One of the unique things about this class is we will go through over ten code examples with you, as well as the Sunshine app. The code for this course is found in two repositories:

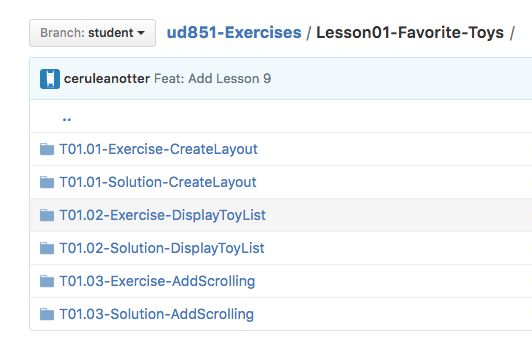
* [**Sunshine Code Repository**](https://github.com/udacity/ud851-Sunshine) ([**ZIP Download**](https://github.com/udacity/ud851-Sunshine/archive/student.zip))
* [**Toy App Repository**](https://github.com/udacity/ud851-Exercises) ([**ZIP Download**](https://github.com/udacity/ud851-Exercises/archive/student.zip))

**Code Steps**

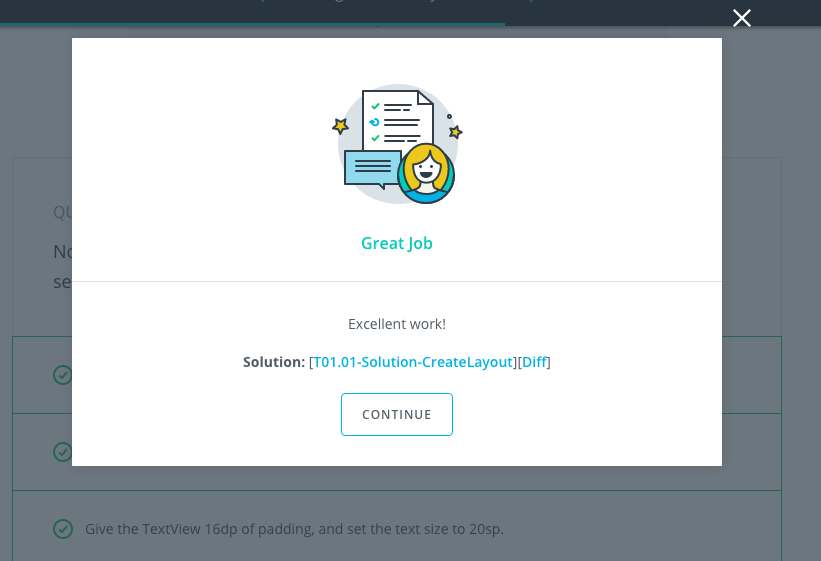
You should download both of the repositories above. As shown in the video, the example repository (also known as the toy app repository) contains folders for each app in the course:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)**

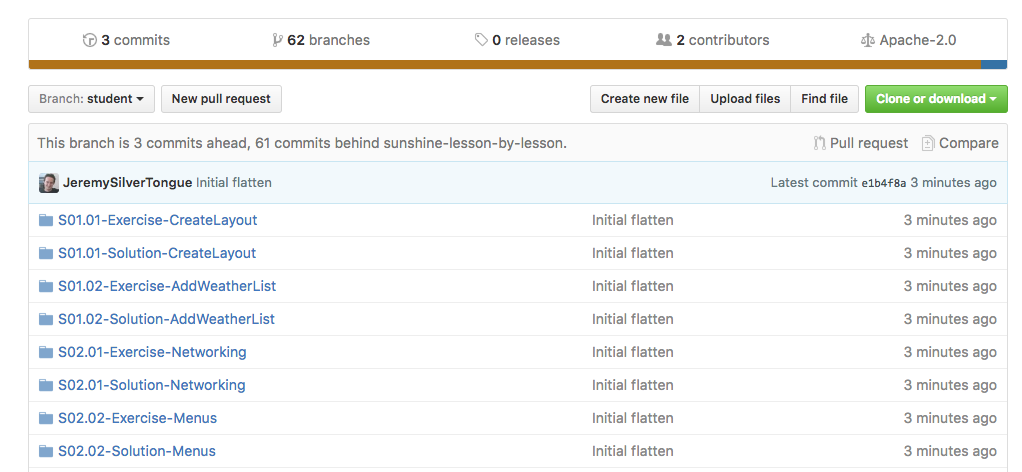
Inside of these folders there are exercises you can do. Each exercise has a starting folder and a solution folder, which contain versions of the app at the start of the exercise and after the exercise.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)**

In addition, when you solve a code step, there are links to GitHub pages which show the code differences between the exercise and solution repositories.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)**

Sunshine is similarly divided up into code steps, with a solution and exercise folder.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/115d08bb-f114-46fa-b693-5c6ce1445c07)**

**Report Issues**

Notice any issues with the repository? File a github issue in the [**Sunshine repository**](https://github.com/udacity/ud851-Sunshine/issues) or in the [**Toy App repository**](https://github.com/udacity/ud851-Exercises/issues).

If you notice a general course issue and want to report it to us, please do! This can include typos, confusing statements, video issues or suggestions. You can report problems by sending an email to android-support(AT)udacity.com. Make sure to have the subject of the email in the following format:

**Developing Android Apps - (Lesson Name) - (Issue description)**

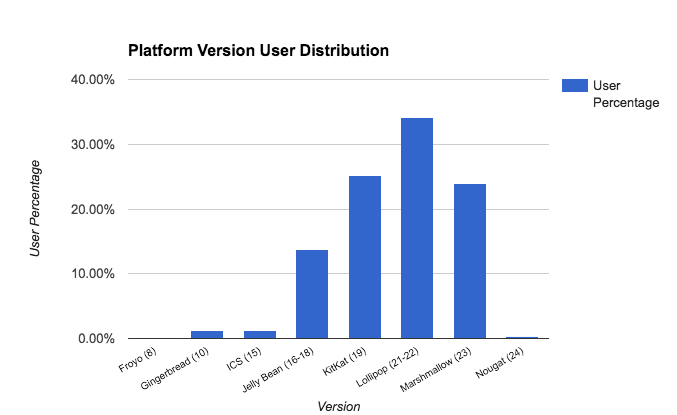
In the body of the email, please include a link to the page you're having trouble with. Taking a screenshot can also help us resolve the issue.

**Android Min and Target Versions**

Android 1.0 launched in 2008, and in just the 8 years since - there have been **13** new major platform releases. By convention, each release is named off of a sugary treat, and the releases are named alphabetically.

On the [**Android Developer site**](https://developer.android.com/about/dashboards/index.html#Platform), we show the relative percentage of active Android devices running a given platform version, also presented in this cool pie chart. Because pie charts are awesome.

For our purposes though, you're really better off looking at it as a histogram.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/5b0aab7f-c63c-4e5d-b336-bdc08985767a)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/5b0aab7f-c63c-4e5d-b336-bdc08985767a)**

If you squint, you can almost see a vaguely bell shaped curve, with the oldest releases on the left, their popularity dropping off as devices are upgraded or replaced. The largest proportion of devices are in the middle, representing devices about 2 years old. The newest platforms, which gain in popularity as new phones are released or updates go out, are on the right.

**Setting minSDK**

The minSDK is the lowest SDK level that your app can run on. You can choose what level of devices to support. Setting the minSDK acts as a filter -- Google Play won’t show your app on devices running an Android platform version lower than your minimum SDK version.

So why not just set the minSDK to 1 and support everyone? Generally, you’ll want to target as many users as you can, but there's a cost associated with supporting older versions - things like creating different execution paths around deprecated or updated APIs, or presenting a different UX to devices with different features. You need to balance the opportunity of expanding your audience, with the cost of supporting those users.

Also remember that each release introduced new APIs and hardware support, so it may not make sense to make your app available to devices that don’t support your minimum feature set. Here are some examples of hardware support and features, tied to releases.

* Home screen widgets (Cupcake)
* Multiple finger tracking (Froyo)
* Tablet (Honeycomb)
* Android Beam (Jellybean)
* Android TV, Auto, Wear (Lollipop)
* Pro Audio (Marshmallow)

**Setting targetSDK**

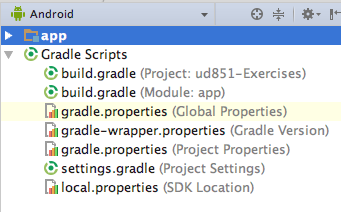
By comparison, the targetSDK is NOT a high pass filter -- it’s used only to declare which platform version you've tested your app on. An app targeted to a certain API or Android version will continue to be forward compatible on future releases -- the platform uses the target SDK values in case a future release makes a significant change to expected behavior, ensuring your app doesn’t break when a user’s phone gets upgraded. `

Android Studio by-default targets the latest release. If you’re developing a new app, there’s really no reason to target anything but the latest Android version, and once your app has been released, make it a point to update your target SDK and test as soon as possible when new platform releases roll out, so your app can take advantage of every new platform optimization and improvement.

# Advanced Build Options

## Gradle in Android Studio

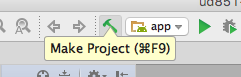
Gradle is the build system of choice for Android Studio. Because of that, there's various functionality available within the platform. When you make a project, there are a few gradle build scripts automatically generated for you, shown below:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

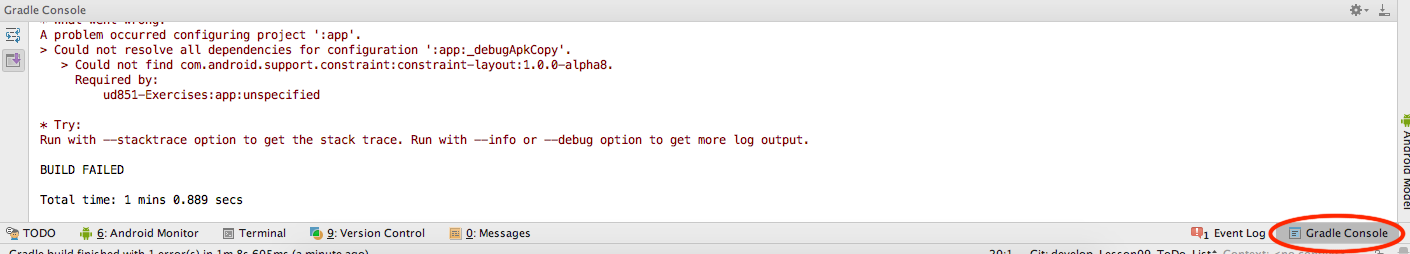
When you press the green **Run** button, if the project has changed, the build scripts will run.

**[[https://d17h27t6h515a5.cloudfront.net/topher/2016/November/582a8833_gradlerunbutton1/gradlerunbutton1.png](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

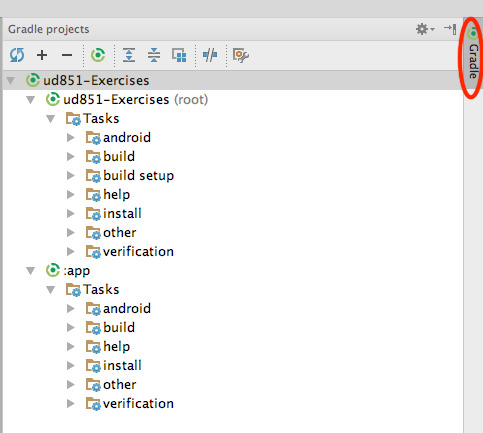
To explicitly run the build scripts, you can press the **Make Project** button:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

To observe logs and build errors for gradle, Android Studio includes a **Gradle Console**.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

A **gradle task** represents a single, atomic piece of work for a build. To see a list of tasks, you can open the tasks window in Android Studio by clicking on the gradle button on the far right. Clicking on the name of the task runs that task.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

## Gradle from the Command Line

You can also run gradle build tasks from the command line if you prefer. To start, you should navigate to the root of your project folder. From there you can run:

./gradlew tasks

This will give you a full list of runnable tasks. You may need to run chmod +x on gradlew before you can run it. See this [**link**](https://developer.android.com/studio/build/index.html#buildCmd) for more detailed instructions. To learn more, you can check out [**Udacity's Gradle course**](https://classroom.udacity.com/courses/ud867/lessons/3968239469/concepts/42836685960923).

## Android Debug Bridge

The other tool that Dan mentioned is **adb** which is short for Android Debug Bridge. ADB is a command line utility included with Android's SDK. You do not need to use adb for this course, but if you enjoy using the command line, you can read up on [**adb**](https://developer.android.com/studio/command-line/adb.html?utm_source=udacity&utm_medium=mooc&utm_term=android&utm_content=adb&utm_campaign=training) and other command line tools.

When you type **adb** into a terminal, you'll also get a helpful list of everything adb can do for you:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/17476436-0b3e-46e3-a858-36d1e35b4491)**

For example, to start your android app from the command line, you could type:

adb shell am start -n com.package.name/com.package.name.ActivityName

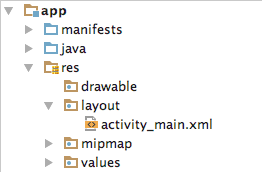
# Android Layouts Primer

You've learned about a lot of new terms that have to do with the Android UI. Here is a quick reference of these important concepts and how they work together. If you have never worked with a language like XML or HTML before and feel confused, you should head over to the [**Android Basics class on Android Layouts**](https://www.udacity.com/course/android-development-for-beginners--ud837)class and take the first lesson.

## Activities and Layouts

An **activity** is a single focused thing that the user can do. Activities are responsible for creating the window that your application uses to draw and receive events from the system. Activities are written in Java, extending from the Activity class.

An activity creates **views** to show the user information, and to let the user interact with the activity. Views are a class in the Android UI framework. They occupy a rectangular area on the screen and are responsible for drawing and handling events. An activity determines what views to create (and where to put them), by reading an XML layout file. These XML files, as Dan mentioned, are stored in the **res folder** inside the folder labeled **layouts**.

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)**

## Layout XML

So what does this XML look like? Here's an example:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="16dp"

tools:context="com.example.android.exampleapp.MainActivity">

<EditText

android:id="@+id/edit\_text\_name\_input"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="@color/colorAccent"

android:hint="Enter your name"

android:padding="4dp"

android:textSize="24sp" />

<TextView

android:id="@+id/text\_view\_name\_display"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

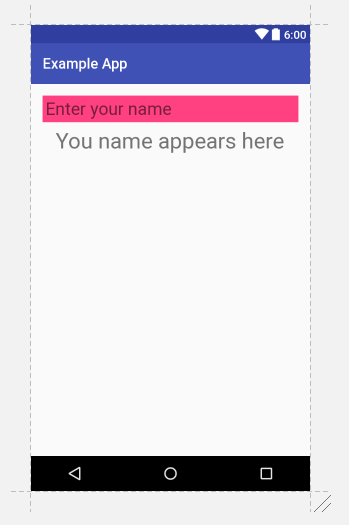
android:layout\_marginTop="8dp"

android:text="Your name appears here"

android:textSize="30sp" />

</LinearLayout>

This looks like the following on a phone screen:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)**

**[Oops! The subtitle should read "Your name appears here"](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)**

## Type of View : UI Components

There are two major categories of views. The first type are UI components that are often interactive. Here are a few examples:

| **Class Name** | **Description** |
| --- | --- |
| [**TextView**](https://developer.android.com/reference/android/widget/TextView.html) | Creates text on the screen; generally non interactive text. |
| [**EditText**](https://developer.android.com/reference/android/widget/EditText.html) | Creates a text input on the screen |
| [**ImageView**](https://developer.android.com/reference/android/widget/ImageView.html) | Creates an image on the screen |
| [**Button**](https://developer.android.com/reference/android/widget/Button.html) | Creates a button on the screen |
| [**Chronometer**](https://developer.android.com/reference/android/widget/Chronometer.html) | Create a simple timer on screen |

The [**android.widget**](https://developer.android.com/reference/android/widget/package-summary.html) package contains a list of most of the UI view classes available to you.

## Type of View : Container View

The second are views called "Layout" or "Container" views. They extend from a class called [**ViewGroup**](https://developer.android.com/reference/android/view/ViewGroup.html). They are primarily responsible for containing a group of views and determining where they are on screen. What do I mean by "containing a group of views?". I mean that a view will be nested inside the tag of another view, like below:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context="com.example.android.exampleapp.MainActivity">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="A"

android:textSize="30sp" />

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:text="B"

android:textSize="30sp" />

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

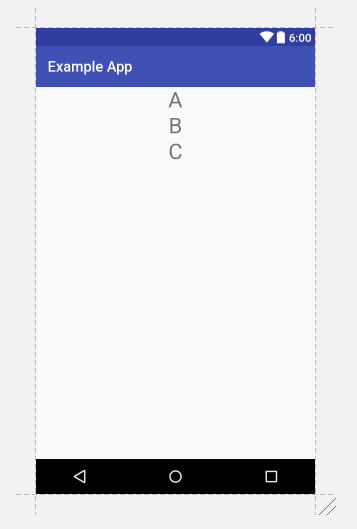
android:layout\_gravity="center"

android:text="C"

android:textSize="30sp" />

</LinearLayout>

This will look like:

**[[](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)**

A few examples of common container views are:

| **Class Name** | **Description** |
| --- | --- |
| [**LinearLayout**](https://developer.android.com/reference/android/widget/LinearLayout.html) | Displays views in a single column or row. |
| [**RelativeLayout**](https://developer.android.com/reference/android/widget/RelativeLayout.html) | Displays views positioned relative to each other and this view. |
| [**FrameLayout**](https://developer.android.com/reference/android/widget/FrameLayout.html) | A ViewGroup meant to contain a single child view. |
| [**ScrollView**](https://developer.android.com/reference/android/widget/ScrollView.html) | A FrameLayout that is designed to let the user scroll through the content in the view. |
| [**ConstraintLayout**](https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html) | This is a newer viewgroup; it positions views in a flexible way. We’ll be exploring constraint layout later in the lesson. |

Note that layout views can be nested in one another, so you can nest a LinearLayout inside of a LinearLayout if you so choose.

## XML Attributes

Views have attributes in XML which control the **properties** of the view. Here’s an example from before:

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

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="16dp"

tools:context="com.example.android.exampleapp.MainActivity">

<EditText

android:id="@+id/edit\_text\_name\_input"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="@color/colorAccent"

android:hint="Enter your name"

android:padding="4dp"

android:textSize="24sp" />

<TextView

android:id="@+id/text\_view\_name\_display"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginTop="8dp"

android:text="Your name appears here"

android:textSize="30sp" />

</LinearLayout>

The properties are things like **textSize** and **padding**. Every view has a handful of properties associated with them, which can be found on their documentation pages. These properties can be set to different values. Properties determine the specifics of how a view looks and interacts. Let's look at a few examples of properties you’ll be using shortly.

### Text

The android:text attribute is an example of a very simple property that can be modified with the xml. In the code above, the portion that says

android:text="Your name appears here"

makes the TextView show the words:

**[[https://d17h27t6h515a5.cloudfront.net/topher/2016/November/5827ad06_image01/image01.png](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)](https://classroom.udacity.com/courses/ud851/lessons/93affc67-3f0b-4f9b-b3a4-a7a26f241a86/concepts/cdbfd437-de24-4903-8f01-37c29427cb38)**

### Width and Height

Some of the most important properties are the width property and height property - those must be defined for every view. Remember that all views occupy a rectangular area on the screen - the width and height are the width and height of that area. You can define this in pixels, or better yet dp (stands for density-independent pixels, we’ll talk a lot more about this in later lessons):

android:layout\_width="200dp"

android:layout\_height="300dp"

For the sake of having a layout be responsive and adjust to different devices, two common values are not numbers at all, but wrap\_content and match\_parent used as shown here:

android:layout\_width="wrap\_content"

android:layout\_height="match\_parent"

**wrap\_content** will shrink the view to wrap whatever is displayed inside the view. For example, if the view is filled with the text “Hello world” then wrap\_content for the width will set the width of the view to be the exact width that the text “Hello world” takes up on the screen.

**match\_parent** will expand the size of the view to be as large as the parent view which it is nested inside of.

### Padding and Margin

padding and layout\_margin are two very similar attributes. Both determine the space around a View. The difference is that padding determines space within the boundaries of the view, and layout\_margindetermines the space outside the boundaries of the view. For a thorough example, check out this video about [**Padding and Margin**](https://classroom.udacity.com/courses/ud837/lessons/4330701752/concepts/42402386170923)

## How do the XML Layouts relate to the Java Activites?

After you create your XML Layout you need to associate it with your activity. This is done in the onCreatemethod of the Activity using the method setContentView. You pass a reference to the layout file as R.layout.name\_of\_layout. For example, if your layout were named **activity\_main.xml** this would look like:

**public** **class** **MainActivity** **extends** **AppCompatActivity** {

@Override

**protected** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

*// other code to setup the activity*

}

*// other code*

}

At this point, you might have two questions, first, what is this R.layout business and second, what is setContentView actually doing?

### The R Class

When your application is compiled the [**R**](https://developer.android.com/reference/android/R.html) class is generated. It creates constants that allow you to dynamically identify the various contents of the res folder, including layouts. To learn more, check out the documentation about [**resources**](https://developer.android.com/guide/topics/resources/accessing-resources.html).

### setContentView

So what is the setContentView method doing? It **inflates the layout**. Essentially what happens is that Android reads your XML file and generates Java objects for each of the tags in your layout file. You can then edit these objects in the Java code by calling methods on the Java objects. We’ll go over what these methods look like and how to access view in your layout files later in the course.

## Additional resources

This was a very quick introduction to the basics of Android layouts. You’ll be learning more about leveraging resources and responsive design in the course. If you’d like a more in-depth introduction to creating XML layouts in Android, check out the first lesson in [**Android for Beginners**](https://www.udacity.com/course/android-development-for-beginners--ud837).

# Update Sunshine Layout Solution

In this step, you modified some of the boiler plate code to create a new layout for Sunshine

## Notes on Solution Code

Most of this step involved massaging the layout code to contain just a simple FrameLayout with a ScrollView, as seen below:

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

android:layout\_width="match\_parent"

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

<ScrollView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView

android:id="@+id/tv\_weather\_data"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:padding="16dp"

android:textSize="20sp" />

</ScrollView>

</FrameLayout>

After removing the ConstraintLayout from your code, it was also safe to remove this dependency:

**compile** 'com.android.support.constraint:constraint-layout:1.0.0-beta3'

### Solution Code

**Solution:** [[**S01.01-Solution-CreateLayout**](https://github.com/udacity/ud851-Sunshine/tree/student/S01.01-Solution-CreateLayout)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S01.01-Exercise-CreateLayout...S01.01-Solution-CreateLayout)]

# Add Scrolling Weather Solution

In this exercise, you added a list of scrolling weather data.

Perhaps the trickiest part of this exercise was using findViewById. findViewById does exactly as the name implies: It gets a view from your XML layout by its ID, and stores it for you in a Java object, like this:

mWeatherTextView = (TextView) findViewById(R.id.tv\_weather\_data);

Once you have this view, you can add everything in your weather array using the [**append**](https://developer.android.com/reference/android/widget/TextView.html#append(java.lang.CharSequence)) method of the TextView class.

Another method you can use is [**setText**](https://developer.android.com/reference/android/widget/TextView.html#setText(int)). The difference between setText and append is that setTextoverwrites what was in the TextView, while append simply adds text onto whatever text was already there. For the solution code and a full diff showing the before and after states, check out the links below.

### Solution Code

**Solution:** [[**S01.02-Solution-AddWeatherList**](https://github.com/udacity/ud851-Sunshine/tree/student/S01.02-Solution-AddWeatherList)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S01.02-Exercise-AddWeatherList...S01.02-Solution-AddWeatherList)]