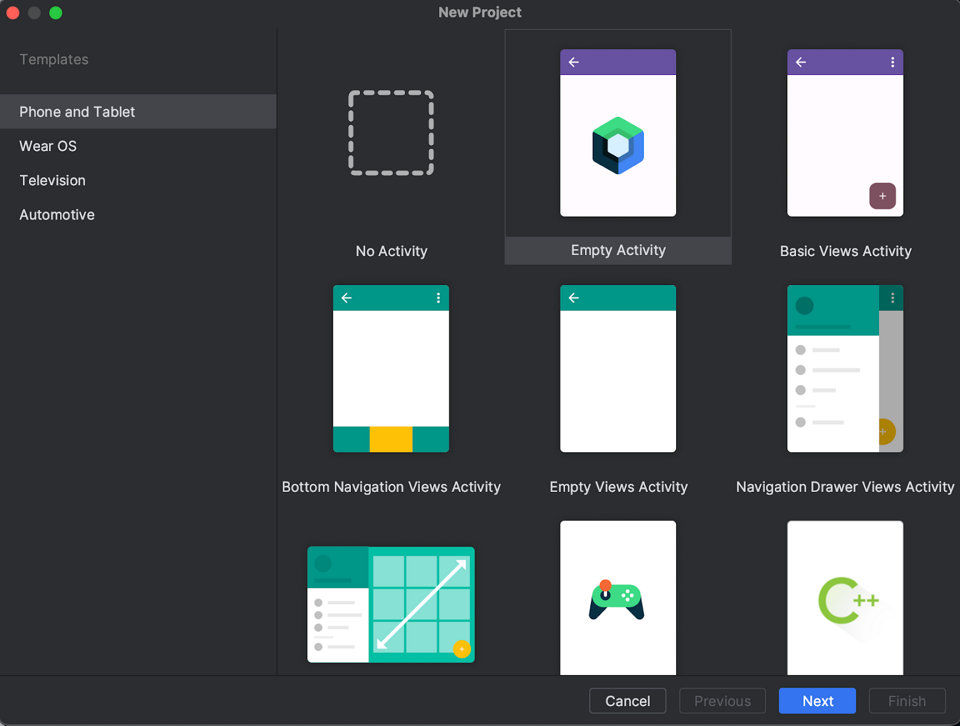
# Objective 1 : Understanding Basics of Jetpack Compose

## [Starting a new Compose project](https://developer.android.com/codelabs/jetpack-compose-basics#1)

To start a new Compose project, open Android Studio.

If you're in the Welcome to Android Studio window, click Start a new Android Studio project. If you already have an Android Studio project open, select File > New > New Project from the menu bar.

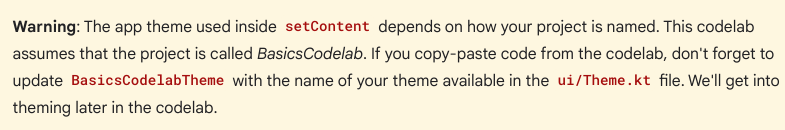
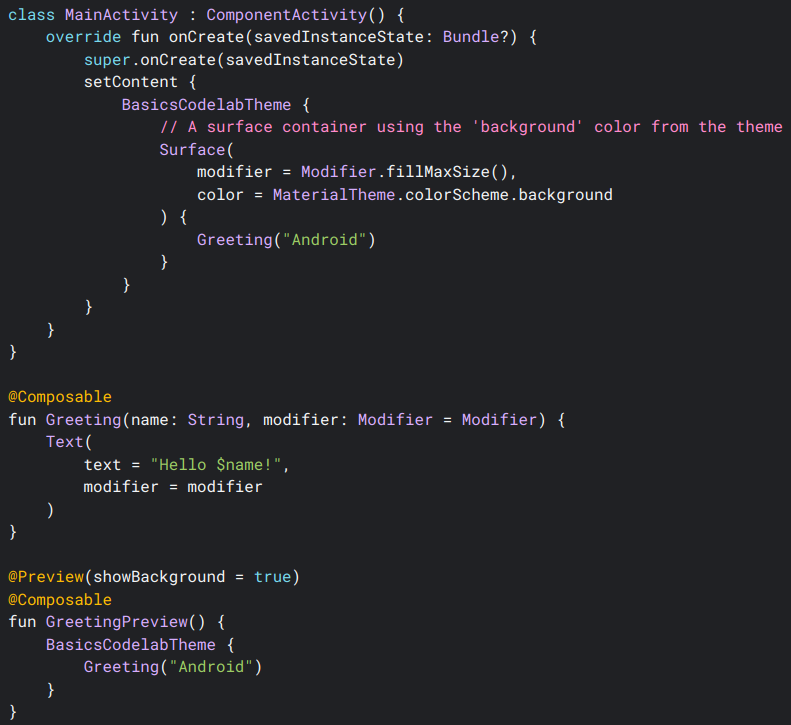
For a new project, choose Empty Activity from the available templates.

Click Next and configure your project as usual, calling it "Basics Codelab". Make sure you select a *minimumSdkVersion* of at least API level 21, which is the minimum API Compose supports.

When choosing the Empty Activity template, the following code is generated for you in your project:

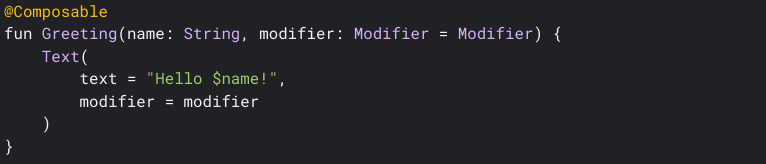
* The project is already configured to use Compose.
* The AndroidManifest.xml file is created.
* The build.gradle.kts and app/build.gradle.kts files contain options and dependencies needed for Compose.

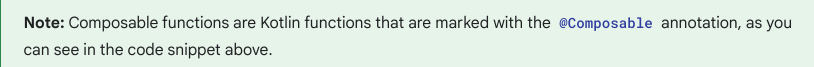
After syncing the project, open MainActivity.kt and check out the code.



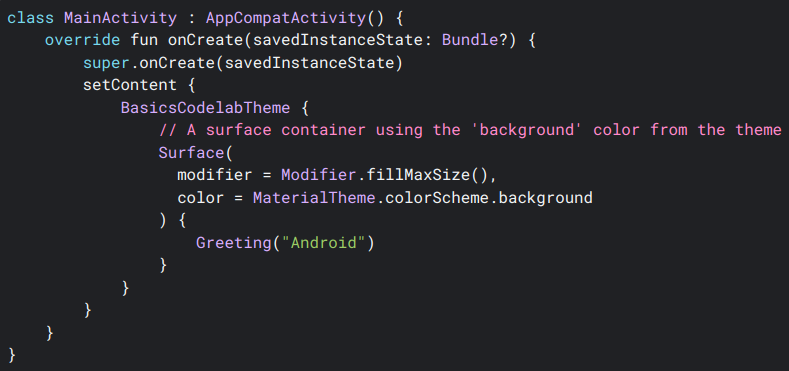
Composable functions

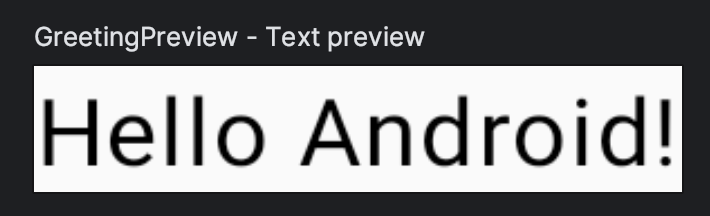
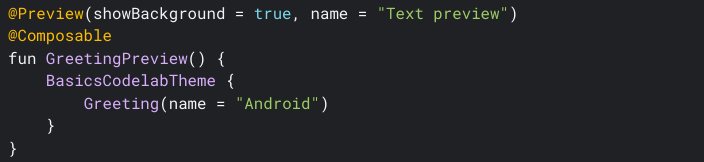
A composable function is a regular function annotated with @Composable. This enables your function to call other @Composable functions within it. You can see how the Greeting function is marked as @Composable. This function will produce a piece of UI hierarchy displaying the given input, String. Text is a composable function provided by the library.



Compose in an Android app

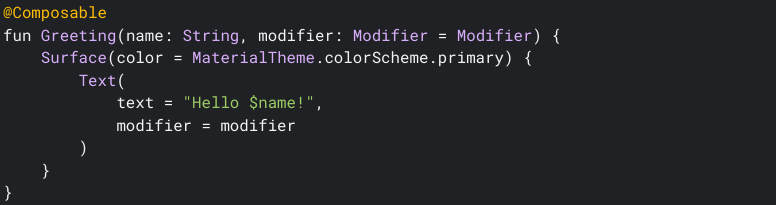
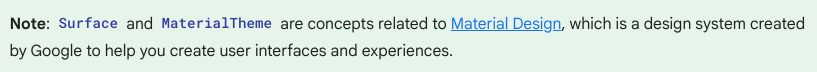
With Compose, an Activity remains the entry point to an Android app. In our project, MainActivity is launched when the user opens the app (as it's specified in the AndroidManifest.xml file). You use setContent to define your layout, but instead of using an XML file as you'd do in the traditional View system, you call Composable functions within it.

To use the Android Studio preview, you just have to mark any parameterless Composable function or functions with default parameters with the @Preview annotation and build your project. You can already see a Preview Composable function in the MainActivity.kt file. You can have multiple previews in the same file and give them names.

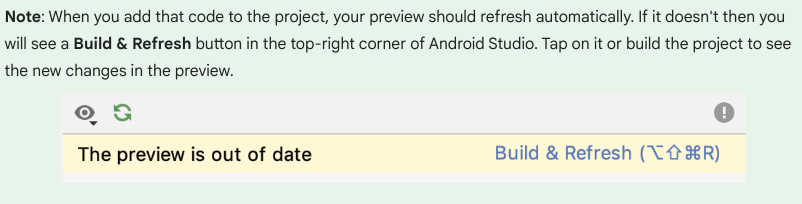


## [Tweaking the UI](https://developer.android.com/codelabs/jetpack-compose-basics#3)

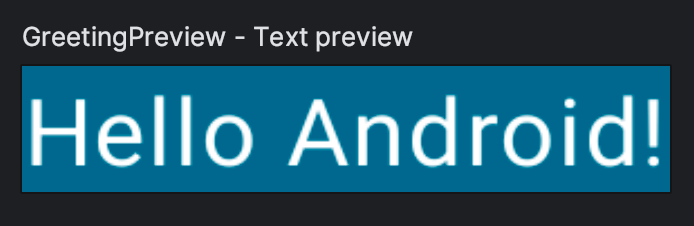
Let's start by setting a different background color for the Greeting. You can do this by wrapping the Text composable with a Surface. Surface takes a color, so use MaterialTheme.colorScheme.primary.



The components nested inside Surface will be drawn on top of that background color.



You can see the new changes in the preview:



You might have missed an important detail: the text is now white. When did we define this?

You didn't! The Material components, such as androidx.compose.material3.Surface, are built to make your experience better by taking care of common features that you probably want in your app, such as choosing an appropriate color for text. We say Material is *opinionated* because it provides good defaults and patterns that are common to most apps. The Material components in Compose are built on top of other foundational components (in androidx.compose.foundation), which are also accessible from your app components in case you need more flexibility.

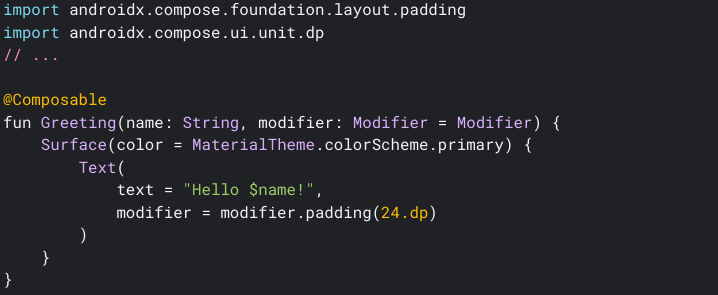
In this case, Surface understands that, when the background is set to the primary color, any text on top of it should use the onPrimary color, which is also defined in the theme. You can learn more about this in the Theming your app section.

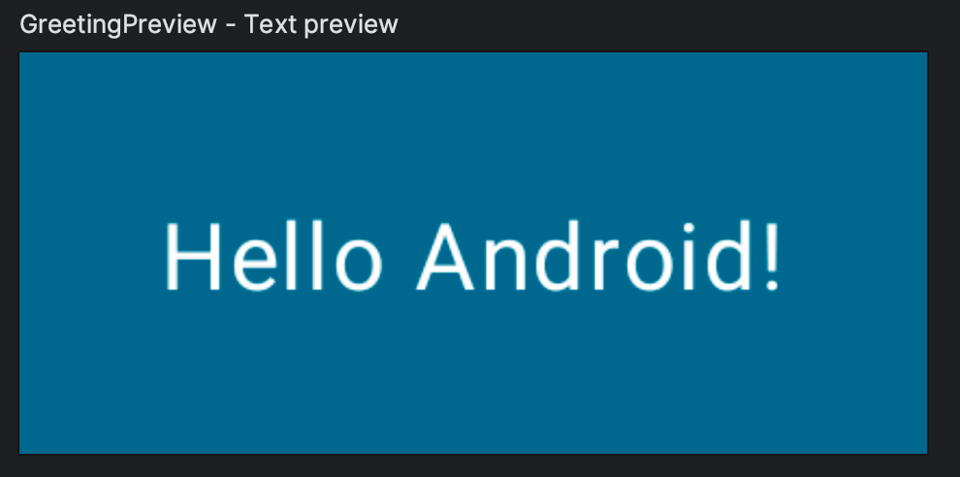
## Modifiers

Most Compose UI elements such as Surface and Text accept an optional modifier parameter. Modifiers tell a UI element how to lay out, display, or behave within its parent layout. You may have already noticed that the Greeting composable already has a default modifier, which is then passed to the Text.

For example, the padding modifier will apply an amount of space around the element it decorates. You can create a padding modifier with Modifier.padding(). You can also add multiple modifiers by chaining them, so in our case we can add the padding modifier to the default one: modifier.padding(24.dp).

Now, add padding to your Text on the screen:



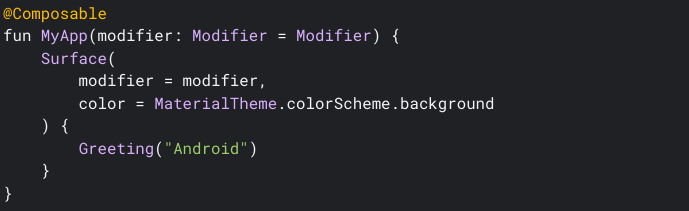


## [Reusing composables](https://developer.android.com/codelabs/jetpack-compose-basics#4)

The more components you add to the UI, the more levels of nesting you create. This can affect readability if a function becomes really large. By making small reusable components it's easy to build up a library of UI elements used in your app. Each one is responsible for one small part of the screen and can be edited independently.

As a best practice, your function should include a Modifier parameter that is assigned an empty Modifier by default. Forward this modifier to the first composable you call inside your function. This way, the calling site can adapt layout instructions and behaviors from outside of your composable function.

Create a Composable called MyApp that includes the greeting.

This lets you clean up the onCreate callback and the preview as you can now reuse the MyApp composable, avoiding code duplication.

In the preview, let's call MyApp and remove the name of the preview.

Your MainActivity.kt file should look like this:

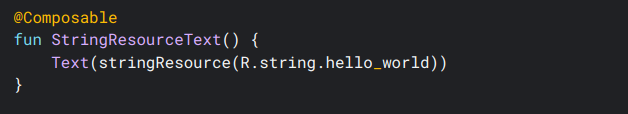
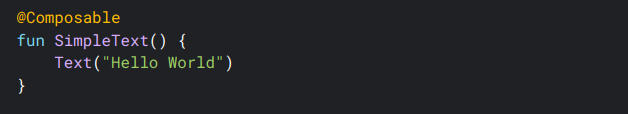


# Objective 2 - Understanding Composable Basic Widgets / Elements / Components

## Text in Compose

Text is a central piece of any UI, and Jetpack Compose makes it easier to display or write text. Compose leverages composition of its building blocks, meaning you don’t need to overwrite properties and methods or extend big classes to have a specific composable design and logic working the way you want.

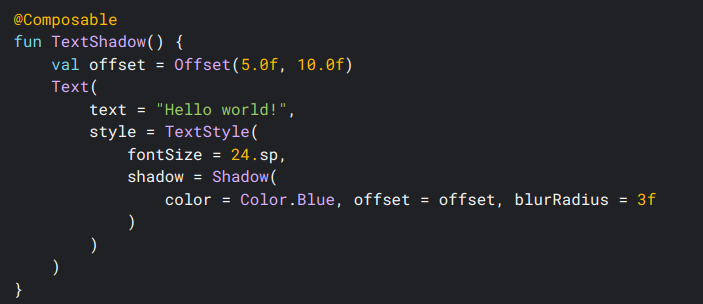
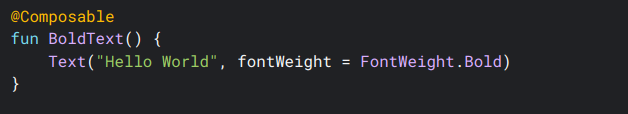
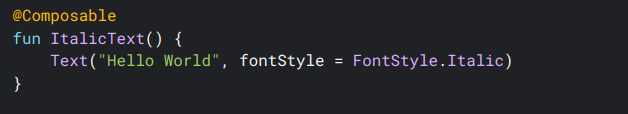
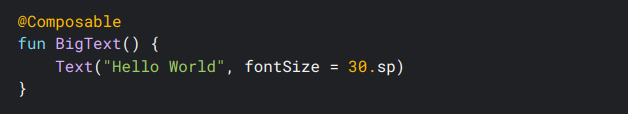
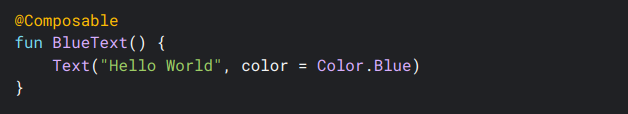
As its base, Compose provides a [BasicText](https://developer.android.com/reference/kotlin/androidx/compose/foundation/text/package-summary#BasicText(androidx.compose.ui.text.AnnotatedString,androidx.compose.ui.Modifier,androidx.compose.ui.text.TextStyle,kotlin.Function1,androidx.compose.ui.text.style.TextOverflow,kotlin.Boolean,kotlin.Int,kotlin.Int,kotlin.collections.Map,androidx.compose.ui.graphics.ColorProducer)) and [BasicTextField](https://developer.android.com/reference/kotlin/androidx/compose/foundation/text/package-summary#BasicTextField(androidx.compose.ui.text.input.TextFieldValue,kotlin.Function1,androidx.compose.ui.Modifier,kotlin.Boolean,kotlin.Boolean,androidx.compose.ui.text.TextStyle,androidx.compose.foundation.text.KeyboardOptions,androidx.compose.foundation.text.KeyboardActions,kotlin.Boolean,kotlin.Int,kotlin.Int,androidx.compose.ui.text.input.VisualTransformation,kotlin.Function1,androidx.compose.foundation.interaction.MutableInteractionSource,androidx.compose.ui.graphics.Brush,kotlin.Function1)), which are the barebones to display text and handle user input. At a higher level, Compose provides [Text](https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary#Text(androidx.compose.ui.text.AnnotatedString,androidx.compose.ui.Modifier,androidx.compose.ui.graphics.Color,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.font.FontStyle,androidx.compose.ui.text.font.FontWeight,androidx.compose.ui.text.font.FontFamily,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextDecoration,androidx.compose.ui.text.style.TextAlign,androidx.compose.ui.unit.TextUnit,androidx.compose.ui.text.style.TextOverflow,kotlin.Boolean,kotlin.Int,kotlin.Int,kotlin.collections.Map,kotlin.Function1,androidx.compose.ui.text.TextStyle)) and [TextField](https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary#TextField(androidx.compose.ui.text.input.TextFieldValue,kotlin.Function1,androidx.compose.ui.Modifier,kotlin.Boolean,kotlin.Boolean,androidx.compose.ui.text.TextStyle,kotlin.Function0,kotlin.Function0,kotlin.Function0,kotlin.Function0,kotlin.Boolean,androidx.compose.ui.text.input.VisualTransformation,androidx.compose.foundation.text.KeyboardOptions,androidx.compose.foundation.text.KeyboardActions,kotlin.Boolean,kotlin.Int,kotlin.Int,androidx.compose.foundation.interaction.MutableInteractionSource,androidx.compose.ui.graphics.Shape,androidx.compose.material.TextFieldColors)), which are composables following Material Design guidelines. It’s recommended to use them as they have the right look and feel for users on Android, and includes other options to simplify their customization without having to write a lot of code.



### Styling Text

The Text composable has multiple optional parameters to style its content. Below, we’ve listed parameters that cover the most common use cases with text. For all the parameters of Text, see the [Compose Text source code](https://cs.android.com/androidx/platform/frameworks/support/+/androidx-main:compose/material/material/src/commonMain/kotlin/androidx/compose/material/Text.kt;l=91).

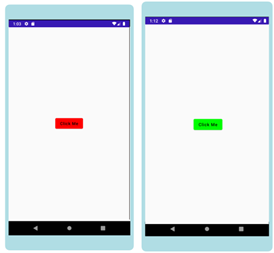
Whenever you set one of these parameters, you’re applying the style to the whole text value. If you need to apply multiple styles within the same line or paragraphs, see the section on [multiple inline styles](https://developer.android.com/jetpack/compose/text/style-text#multiple-styles).



# Objective 4 - Practice Tasks

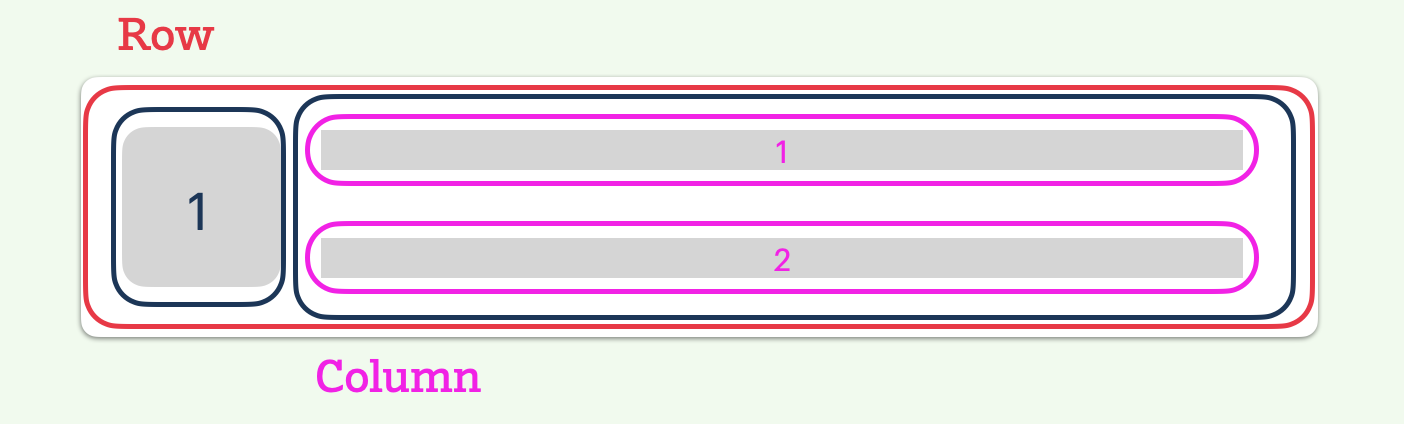
## Task 1 : Managing State

Change the color of the button on click using the state. Create one button, make it the center of the screen. Put the initial color red, then onClick event change its color to green.



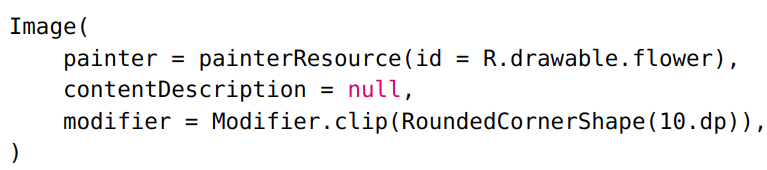
## Task 2 : Layouts Practice

Create the task imitating the following screenshot.



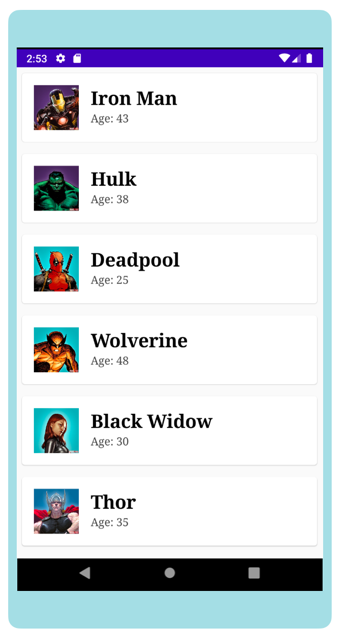
Components:

1. Row
2. Column (Stretchable, must contain the remaining size after image)
3. Image (RoundedCorner)
4. Two TextFileds (denoting Name and Phone Number)

Code to Clip the image in RoundedCorners Shape:  


## Task 3 : Creating multiple rows.

Use task 2, to create the following screen.



## Task 4 : Creating Grid using LazyGrid.

Create the following task using LazyGrid Layout

