Create your first Android app

1. Before you begin

Install Android Studio on your computer if you haven't done so already. Check that your computer meets the system requirements required for running Android Studio (located at the bottom of the download page). If you need more detailed instructions on the setup process, refer to the Download and install Android Studio codelab.

In this codelab, you create your first Android app with a project template provided by Android Studio. You use Kotlin and Jetpack Compose to customize your app. Note that Android Studio gets updated and sometimes the UI changes so it is okay if your Android Studio looks a little different than the screenshots in this codelab.

Prerequisites

- Basic Kotlin knowledge

What you'll need

- The latest version of Android Studio

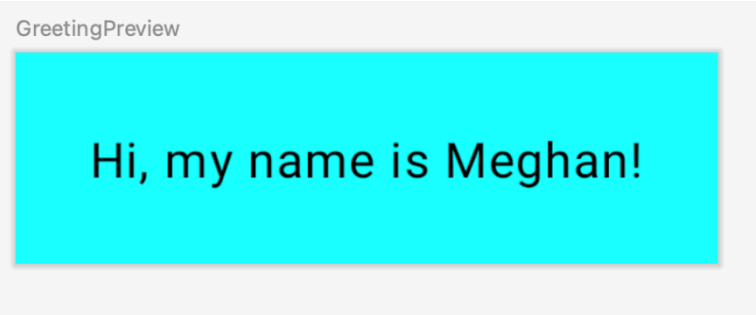
What you'll learn

- How to create an Android App with Android Studio
- How to run apps with the Preview tool in Android Studio
- How to update text with Kotlin
- How to update a User Interface (UI) with Jetpack Compose
- How to see a preview of your app with Preview in Jetpack Compose

What you'll build

- An app that lets you customize your introduction!

Here's what the app looks like when you complete this codelab (except it will be customized with your name!):



What you'll need

- A computer with Android Studio installed.

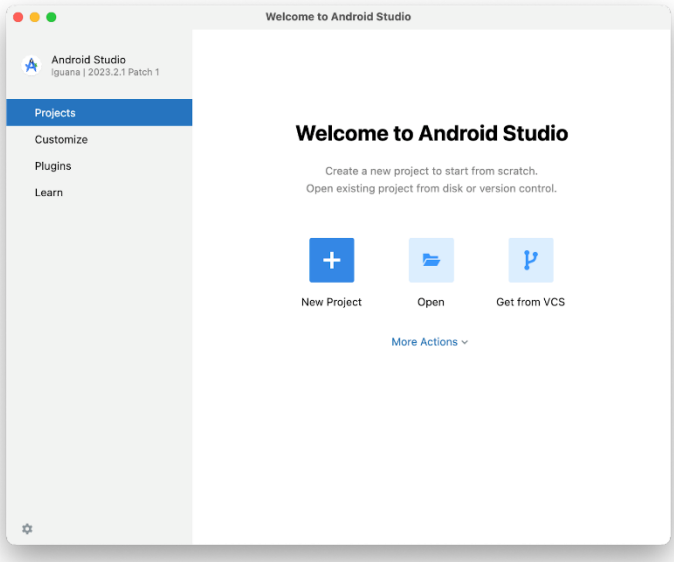
2. Create a project using the template

In this codelab, you create an Android app with the **Empty Activity** project template provided by Android Studio.

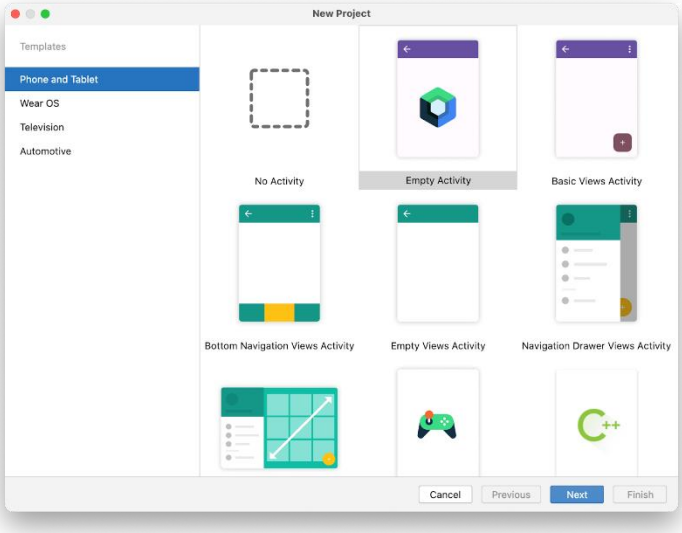
To create a project in Android Studio:

1. Double click the Android Studio icon to launch Android Studio.

2. In the **Welcome to Android Studio** dialog, click **New Project**.



The **New Project** window opens with a list of templates provided by Android Studio.



In Android Studio, a project template is an Android project that provides the blueprint for a certain type of app. Templates create the structure of the project and the files needed for Android Studio to build your project. The template that you choose provides starter code to get you going faster.

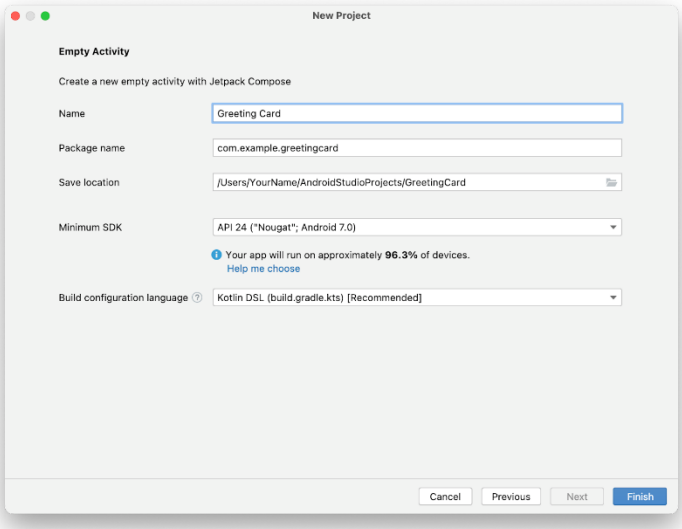
3. Make sure the **Phone and Tablet** tab is selected.
4. Click the **Empty Activity** template to select it as the template for your project. The **Empty Activity** template is the template to create a simple project that you can use to build a Compose app. It has a single screen and displays the text "`Hello Android!`".
5. Click **Next**. The **New Project** dialog opens. This has some fields to configure your project.
6. Configure your project as follows:

The **Name** field is used to enter the name of your project, for this codelab type "Greeting Card".

Leave the **Package name** field as is. This is how your files will be organized in the file structure. In this case, the package name will be `com.example.greetingcard`.

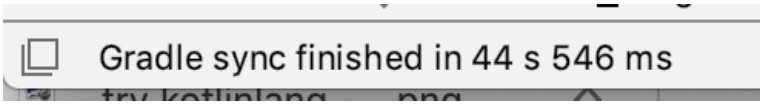
Leave the **Save location** field as is. It contains the location where all the files related to your project are saved. Take a note of where that is on your computer so that you can find your files.

Select **API 24: Android 7.0 (Nougat)** from the menu in the **Minimum SDK** field. Minimum SDK (https://developer.android.com/guide/topics/manifest/uses-sdk-element) indicates the minimum version of Android that your app can run on.

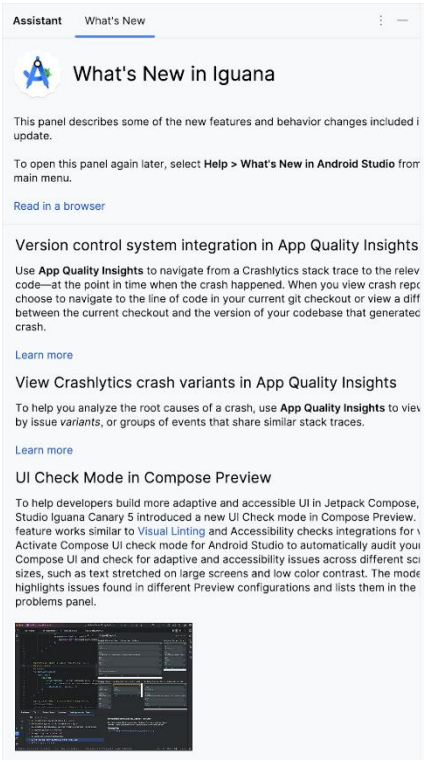


7. Click **Finish**. This may take a while - this is a great time to get a cup of tea! While Android Studio is setting up, a progress bar and message indicates whether Android Studio is still setting up your project. It may look like this:

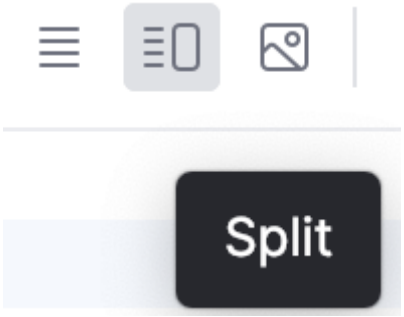
A message that looks similar to this informs you when the project set up is created.



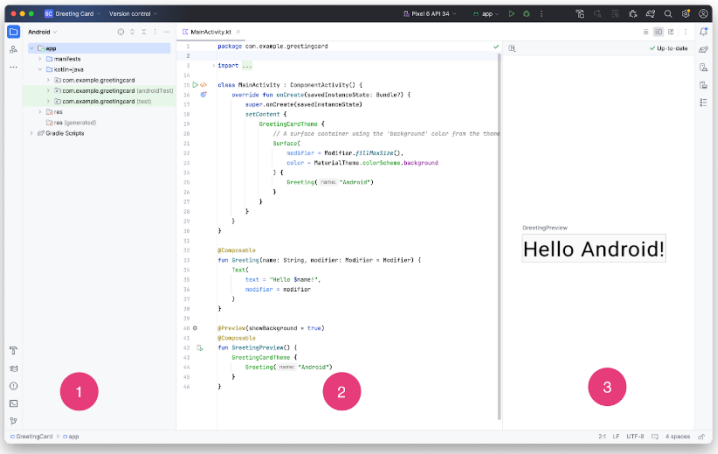
8. You may see a **What's New** pane which contains updates on new features in Android Studio. Close it for now.



9. Click **Split** on the top right of Android Studio, this allows you to view both code and design. You can also click **Code** to view code only or click **Design** to view design only.

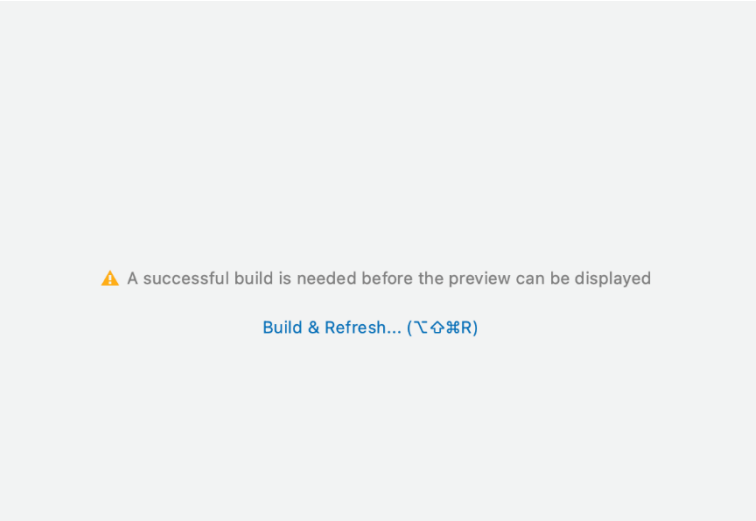


After pressing **Split** you should see three areas:



- The **Project** view (1) shows the files and folders of your project
- The **Code** view (2) is where you edit code
- The **Design** view (3) is where you preview what your app looks like

In the **Design** view, you may see a blank pane with this text:

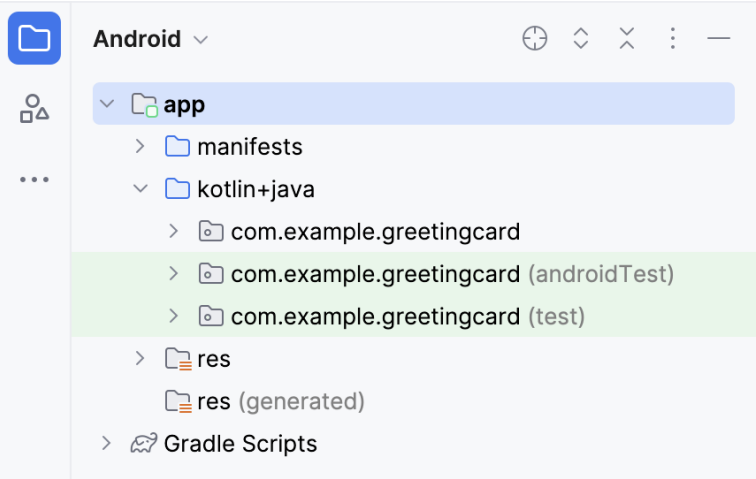


10. Click **Build & Refresh**. It may take a while to build but when it is done the preview shows a text box that says "**Hello Android!**". Empty Compose activity contains all the code necessary to create this app.

3. Find project files

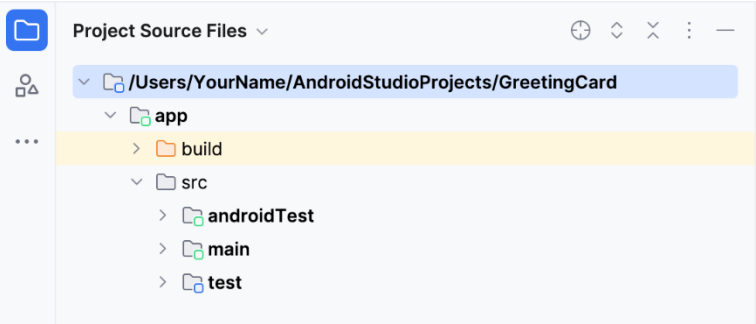
In this section you will continue to explore Android Studio by becoming familiar with the file structure.

1. In Android Studio, take a look at the **Project** tab. The **Project** tab shows the files and folders of your project. When you were setting up your project the package name was **com.example.greetingcard**. You can see that package right here in the **Project** tab. A package is basically a folder where code is located. Android Studio organizes the project in a directory structure made up of a set of packages.
2. If necessary, select **Android** from the drop-down menu in the **Project** tab.



This is the standard view and organization of files that you use. It's useful when you write code for your project because you can easily access the files you will be working on in your app. However, if you look at the files in a file browser, such as Finder or Windows Explorer, the file hierarchy is organized very differently.

3. Select **Project Source Files** from the drop-down menu. You can now browse the files in the same way as in any file browser.



4. Select **Android** again to switch back to the previous view. You use the **Android** view for this course. If your file structure ever looks strange, check to make sure you're still in **Android** view.

4. Update the text

Now that you have gotten to know Android Studio, it's time to start making your greeting card!

Look at the **Code** view of the `MainActivity.kt` file. Notice there are some automatically generated functions in this code, specifically the `onCreate()` and the `setContent()` functions.

Note: Remember that a function is a segment of a program that performs a specific task.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GreetingCardTheme {
                // A surface container using the 'background'
color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color =
MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}
```

The `onCreate()` function is the entry point to this Android app and calls other functions to build the user interface. In Kotlin programs, the `main()` function is the entry point/starting point of execution. In Android apps, the `onCreate()` function fills that role.

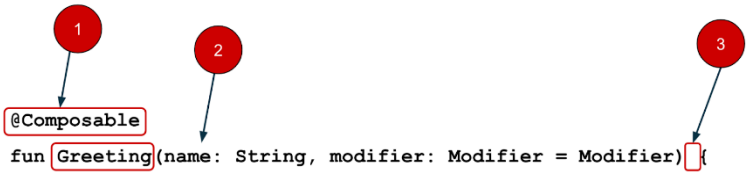
The `setContent()` function within the `onCreate()` function is used to define your layout through composable functions. All functions marked with the `@Composable` annotation can be called from the `setContent()` function or from other Composable functions. The annotation tells the Kotlin compiler that this function is used by Jetpack Compose to generate the UI.

Note: The compiler takes the Kotlin code you wrote, looks at it line by line, and translates it into something that the computer can understand. This process is called compiling your code.

Next, look at the `Greeting()` function. The `Greeting()` function is a Composable function, notice the `@Composable` annotation above it. This Composable function takes some input and generates what's shown on the screen.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

You've learned about functions before (if you need a refresher, visit the Create and use functions in Kotlin codelab), but there are a few differences with composable functions.



1. You add the `@Composable` annotation before the function.
2. `@Composable` function names are capitalized.
3. `@Composable` functions can't return anything.

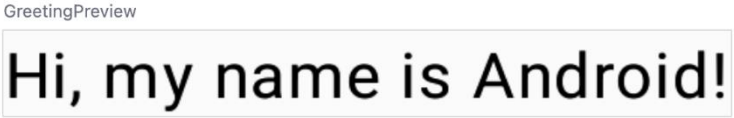
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

Right now the `Greeting()` function takes in a name and displays `Hello` to that person.

1. Update the `Greeting()` function to introduce yourself instead of saying "Hello":

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hi, my name is $name!",
        modifier = modifier
    )
}
```

2. Android should automatically update the preview.



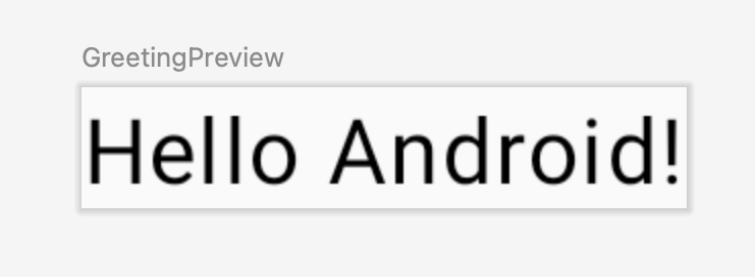
Great! You changed the text, but it introduces you as Android, which is probably not your name. Next, you will personalize it to introduce you with your name!

The `GreetingPreview()` function is a cool feature that lets you see what your composable looks like without having to build your entire app. To enable a preview of a

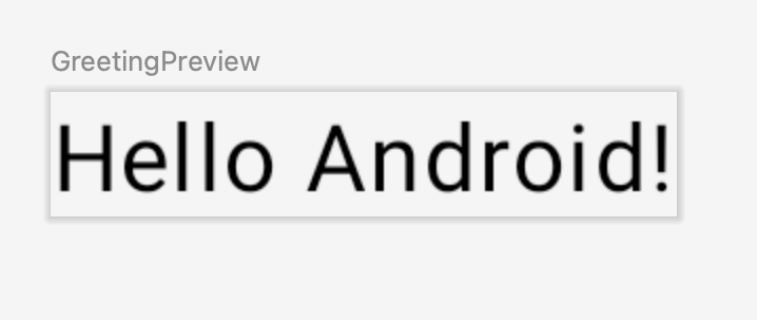
composable, annotate with `@Composable` and `@Preview`. The `@Preview` annotation tells Android Studio that this composable should be shown in the design view of this file.

As you can see, the `@Preview` annotation takes in a parameter called `showBackground`. If `showBackground` is set to **true**, it will add a background to your composable preview.

Since Android Studio by default uses a light theme for the editor, it can be hard to see the difference between `showBackground = true` and `showBackground = false`. However, this is an example of what the difference looks like. Notice the white background on the image set to true.



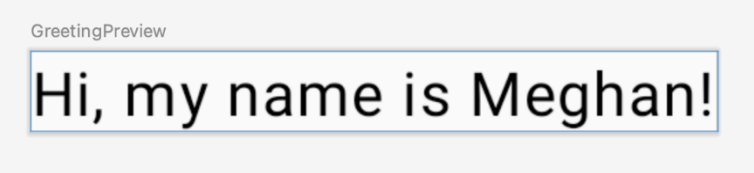
showBackground = true



showBackground = false

3. Update the `GreetingPreview()` function with your name. Then rebuild and check out your personalized greeting card!

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting("Meghan")
    }
}
```

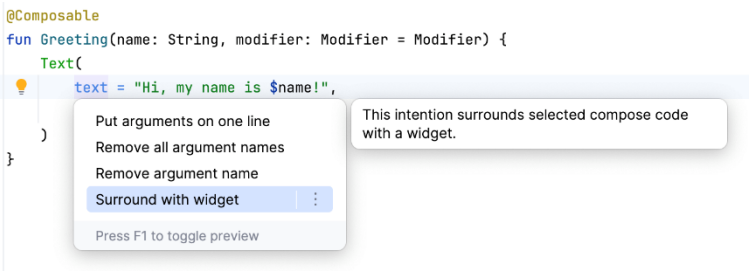


5. Change the background color

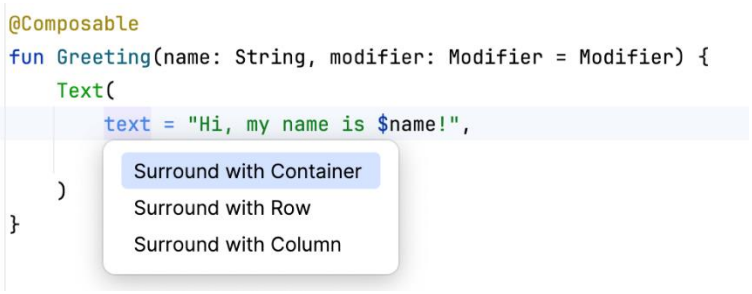
Now you have the introduction text, but it's a little boring! In this section, you learn to change the background color.

To set a different background color for your introduction, you'll need to surround your text with a `Surface`. A `Surface` is a container that represents a section of UI where you can alter the appearance, such as the background color or border.

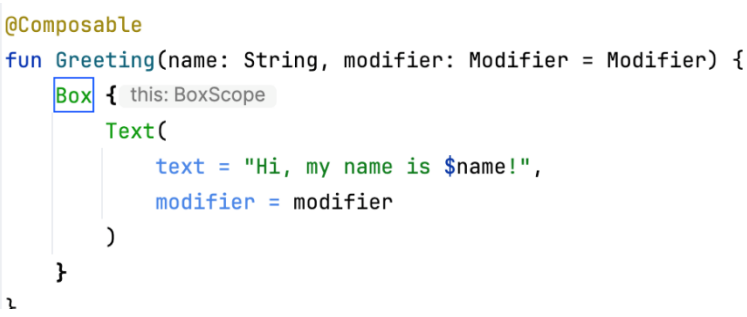
1. To surround the text with a `Surface`, highlight the line of text, press (`Alt+Enter` for Windows or `Option+Enter` on Mac), and then select **Surround with widget**.



2. Choose **Surround with Container**.



The default container it will give you is `Box`, but you can change this to another container type. You will learn about `Box` layout later in the course.



3. Delete `Box` and type `Surface()` instead.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface() {
        Text(
            text = "Hi, my name is $name!",
```

```
            modifier = modifier
        )
    }
}
```

4. To the `Surface` container add a `color` parameter, set it to `Color`.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

5. When you type `Color` you may notice that it is red, which means Android Studio is not able to resolve this. To solve this scroll to the top of the file where it says import and press the three buttons.

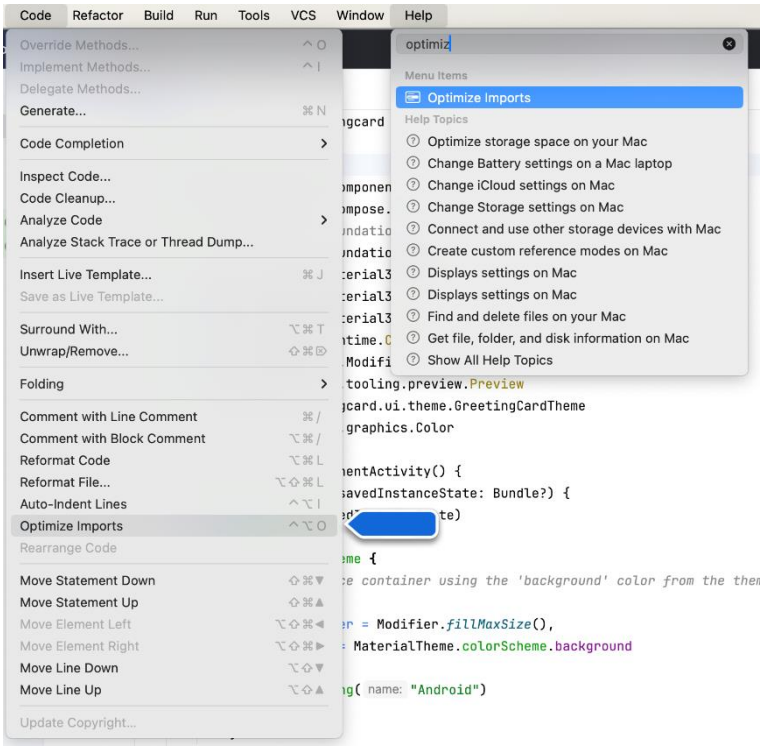
6. Add this statement to the bottom of the list of imports.

```
import androidx.compose.ui.graphics.Color
```

The full list of imports will look similar to this.

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.example.greetingcard.ui.theme.GreetingCardTheme
import androidx.compose.ui.graphics.Color
```

7. In your code, the best practice is to keep your imports listed alphabetically and remove unused imports. To do this press **Help** on the top toolbar, type in **optimize imports**, and click on **Optimize Imports**.



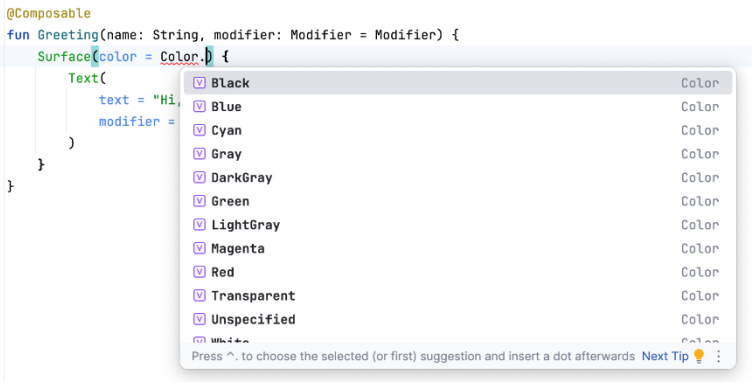
You could open the **Optimize Imports** directly from the menu: **Code** > **Optimize Imports**. Using Help's search option will help you locate a menu item if you don't remember where it is.

The full list of imports will now look like this:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import com.example.greetingcard.ui.theme.GreetingCardTheme
```

8. Notice that the Color that you typed in the Surface parentheses has switched from being red to being underlined in red. To fix that, add a period after it. You will see a pop-up showing different color options.

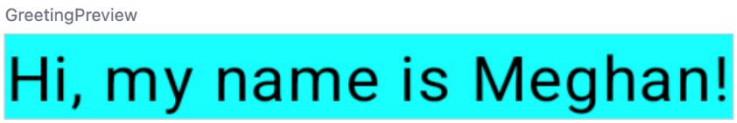
This is one of the cool features in Android Studio, it is intelligent and will help you out when it can. In this case it knows you are wanting to specify a color so it will suggest different colors.



9. Choose a color for your surface. This codelab uses **Cyan**, but you can choose your favorite!

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
   Surface(color = Color.Cyan) {
       Text(
           text = "Hi, my name is $name!",
           modifier = modifier
       )
   }
}
```

10. Notice the updated preview.



6. Add padding

Now your text has a background color, next you will add some space (padding) around the text.

A `Modifier` is used to augment or decorate a composable. One modifier you can use is the `padding` modifier, which adds space around the element (in this case, adding space around the text). This is accomplished by using the `Modifier.padding()` function.

Every composable should have an optional parameter of the type `Modifier`. This should be the first optional parameter.

1. Add a padding to the `modifier` with a size of `24.dp`.

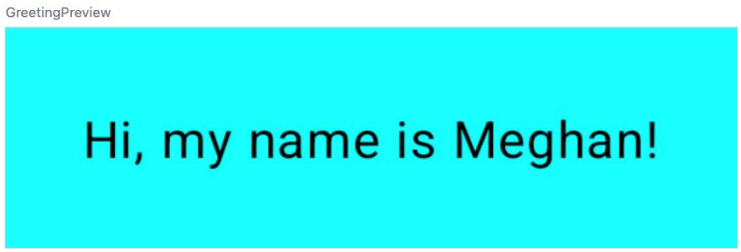
Note: You learn more about density-independent pixels (`dp`) in the next pathway, but refer to Layout – Material Design 3 (https://m3.material.io/foundations/layout/understanding-layout/spacing#abccd6ce-1092-4ad0-9351-de75aeae0edf) article if you want to read more now.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Cyan) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier.padding(24.dp)
        )
    }
}
```

2. Add these imports to the import statement section.

Make sure to use **Optimize Imports** to alphabetize the new imports.

```
import androidx.compose.ui.unit.dp
import androidx.compose.foundation.layout.padding
```



Well congratulations - you built your first Android app in Compose! This is a pretty huge accomplishment. Take some time to play around with different colors and text, make it your own!

7. Review the solution code

Code snippet for review

```
package com.example.greetingcard

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.greetingcard.ui.theme.GreetingCardTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GreetingCardTheme {
                // A surface container using the 'background'
color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color =
MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Cyan) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier.padding(24.dp)
        )
    }
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting("Meghan")
    }
}
```

8. Conclusion

You learned about Android Studio and built your first Android app with Compose, great job!

This codelab is part of the Android Basics with Compose course. To learn how to run your app on the emulator or a physical device, check out the next codelabs in this pathway (https://developer.android.com/courses/pathways/android-basics-compose-unit-1-pathway-2).

Summary

- To create a new project: open Android Studio, click **New Project** > **Empty Activity** > **Next**, enter a name for your project and then configure its settings.
- To see how your app looks, use the **Preview** pane.
- Composable functions are like regular functions with a few differences: functions names are capitalized, you add the `@Composable` annotation before the function, `@Composable` functions can't return anything.
- A `Modifier` is used to augment or decorate your composable.

Learn more

Meet Android Studio	https://developer.android.com/studio/intro
Projects overview	https://developer.android.com/studio/projects
Create a project	https://developer.android.com/studio/projects/create-project
Add code from a template	https://developer.android.com/studio/projects/templates
Jetpack Compose	https://developer.android.com/jetpack/compose
Padding – Material Design 3	https://m3.material.io/foundations/layout/understanding-layout/spacing#64eb2223-f5e8-4d2a-9edc-9e3a7002220a