Develop mobile application

Create your first Android app

1. Before you begin

Setup the Environment by installing:

I. Java Development Kit (JDK)
II. Android Studio

In this section, you will create your first Android app with a project template provided by Android Studio. You will use Kotlin and Jetpack Compose to customize your app.

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!

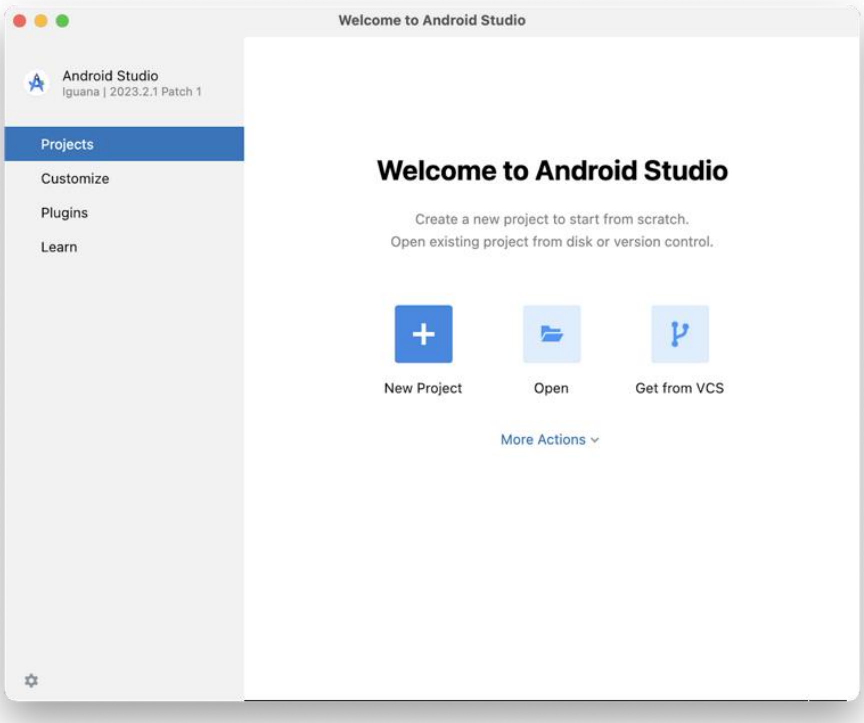
2. Create a project using the template

You will create an Android app with the Empty Activity project template provided by Android Studio.

To create a project in Android Studio:

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

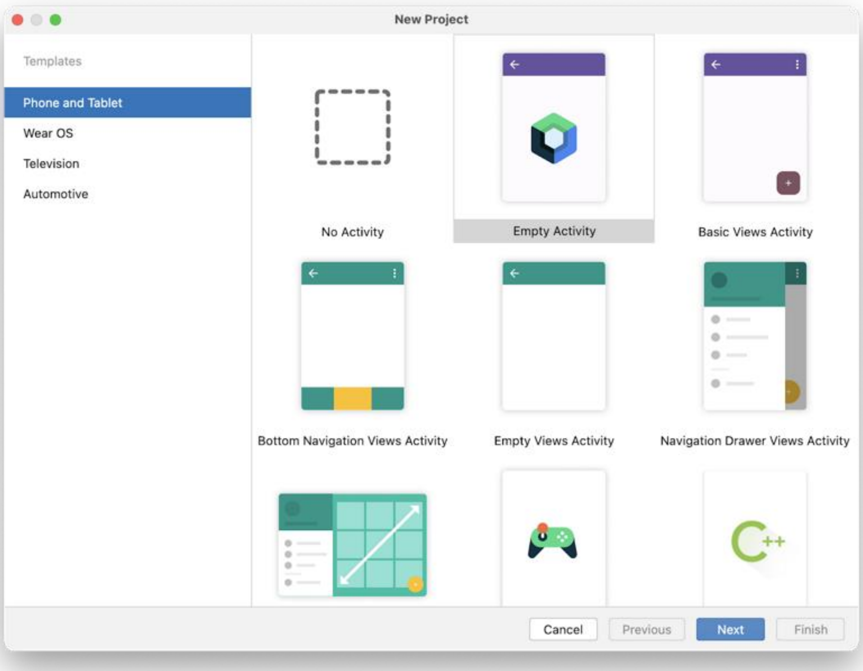
II. In the Welcome to Android Studio dialog, click New Project.



Welcome to Android Studio

Create a new project to start from scratch.
Open existing project from disk or version control.

III. 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:

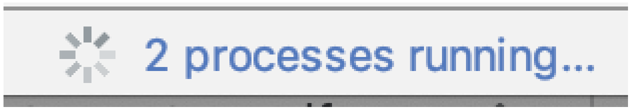
The Name field is used to enter the name of your project, for this codelab type "Greeting Card".

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.

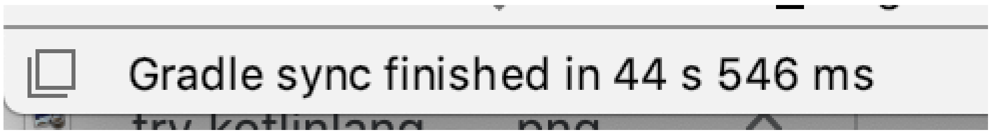
Select API 24: Android 7.0 (Nougat) from the menu in the Minimum SDK field. Minimum SDK indicates the minimum version of Android that your app can run on.

7. Click Finish. This may take a while. It may look like this:



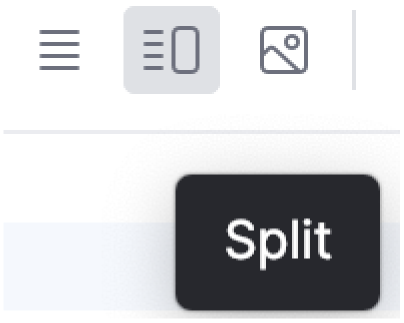
2 processes running...

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

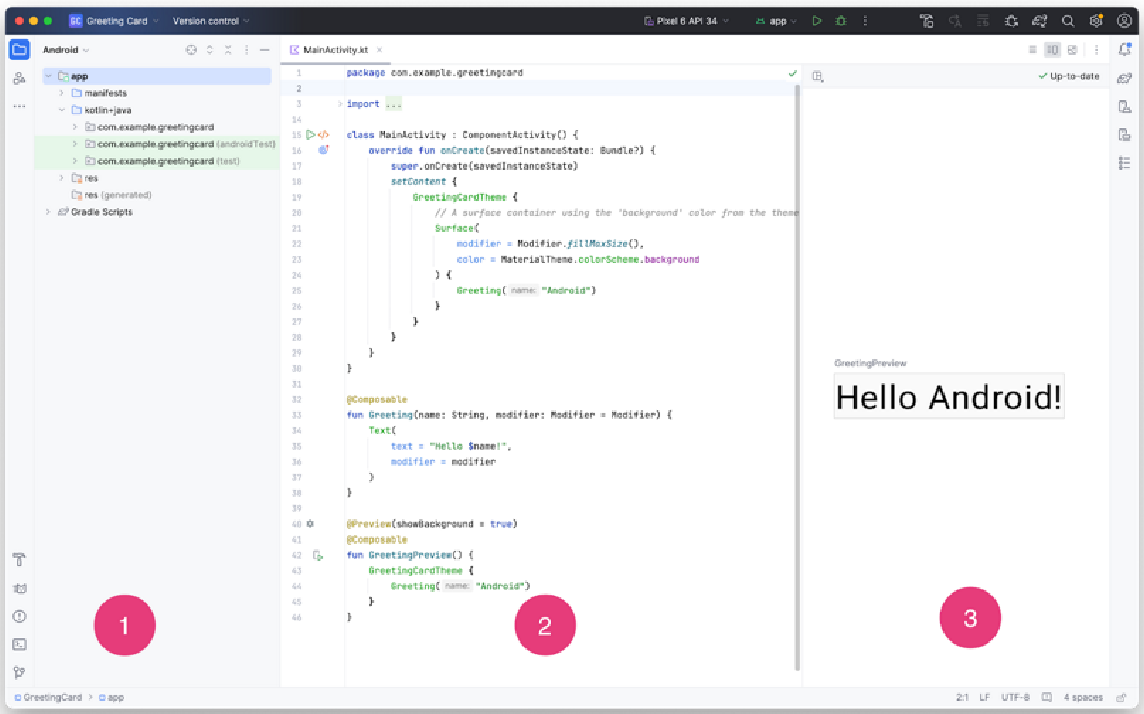


Gradle sync finished in 44 s 546 ms

8. 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.



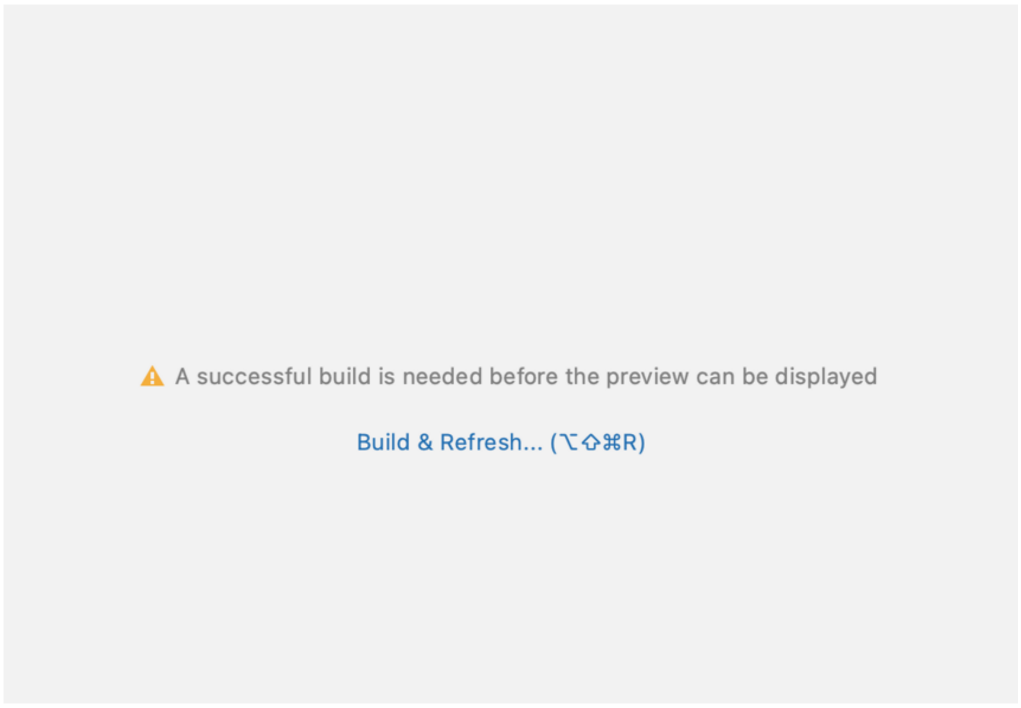
After pressing Split you should see three areas:



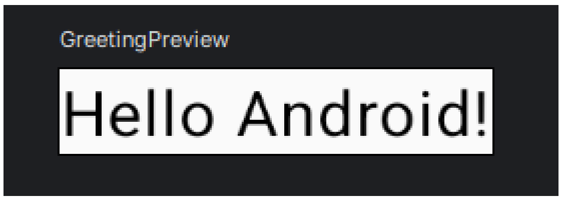
```kotlin
package com.example.greetingcard

import ...

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( name: "Android")
                }
            }
        }
    }
}

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

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

- 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:


⚠ A successful build is needed before the preview can be displayed

9. 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.

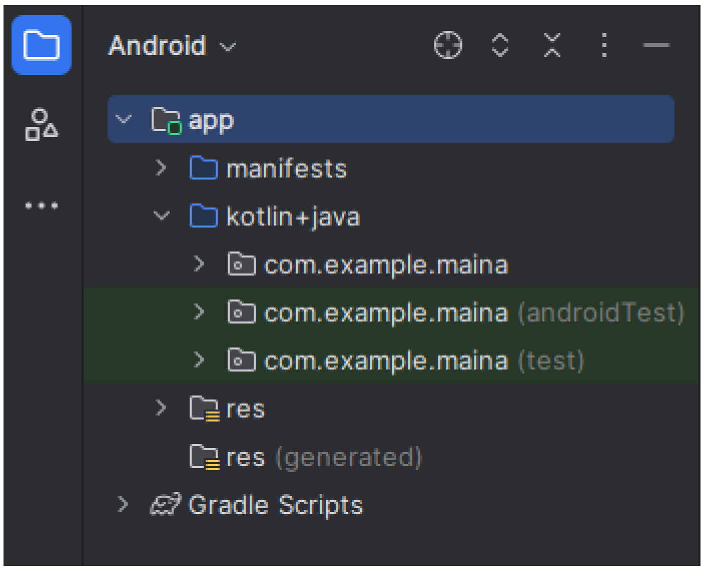


Hello Android!

3. Find project files

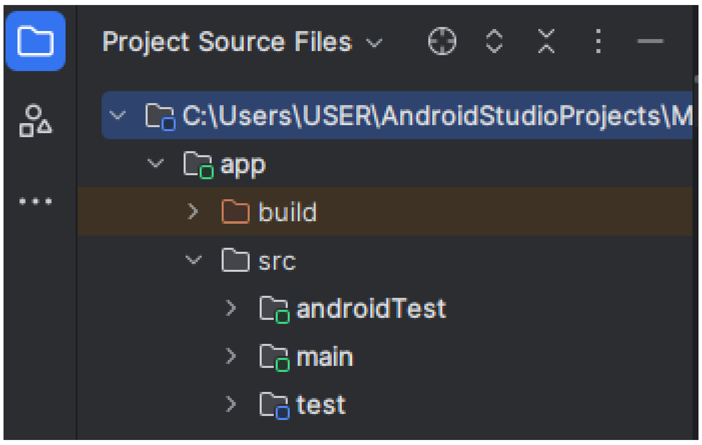
In this section you will continue to explore Android Studio by becoming familiar with the file structure.

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.yourname**. 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.

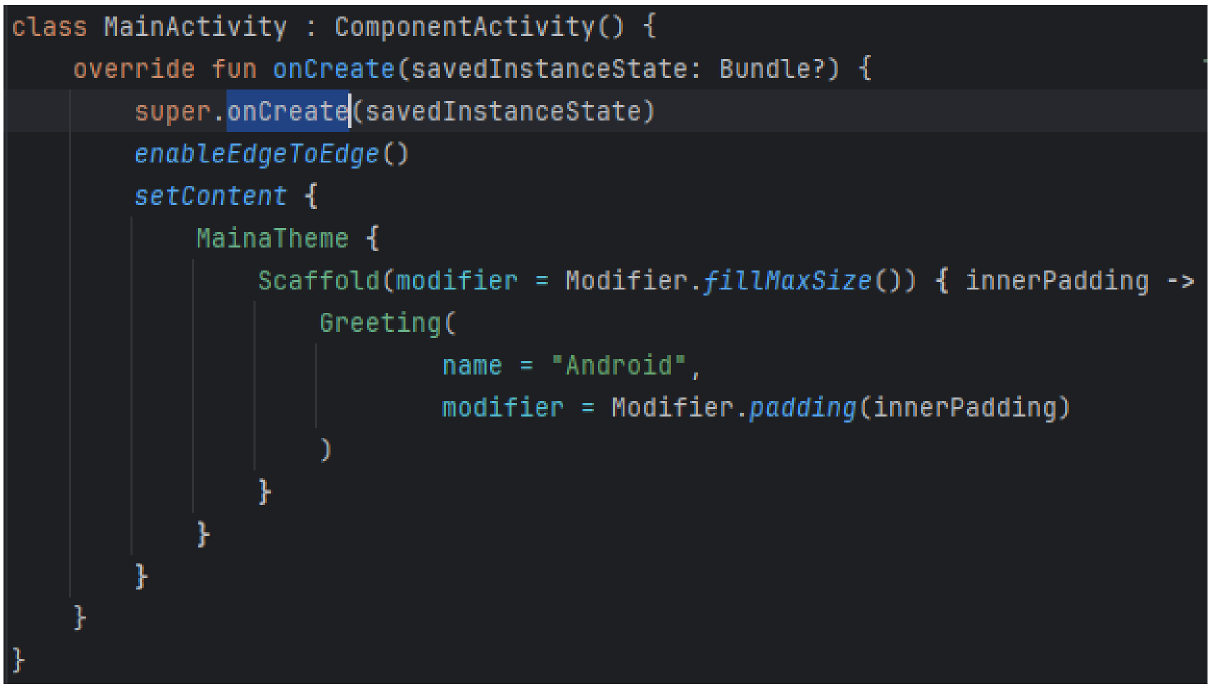
3. Select **Project Source Files** from the drop-down menu. You can now browse the files in the same way as in any file browser.



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

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.



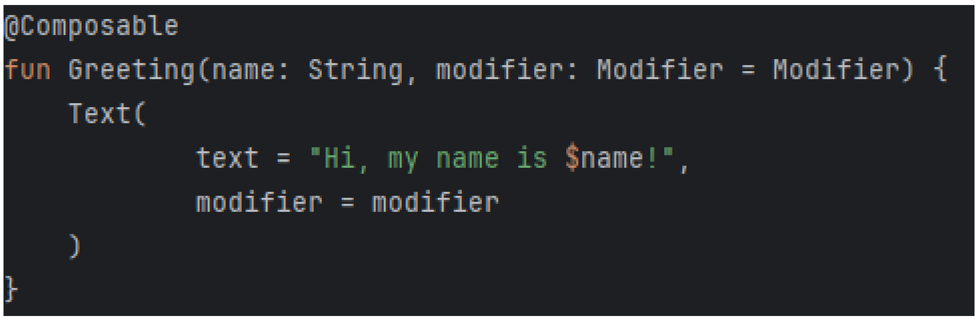
```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MainaTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Greeting(
                            name = "Android",
                            modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

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.

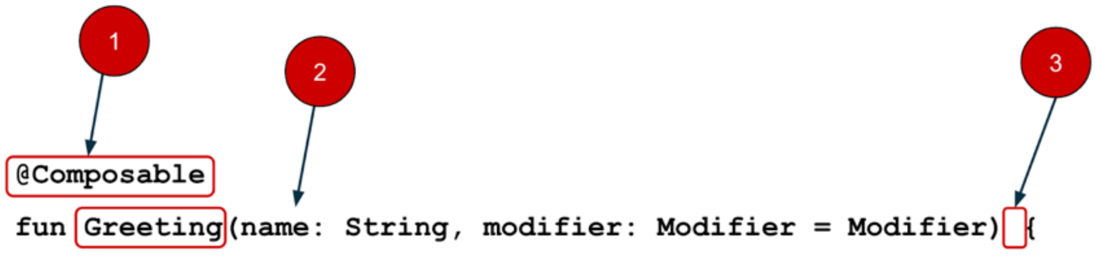
The Greeting() function is a Composable function, notice the @Composable annotation above it. This Composable function takes some input and generates what's shown on the screen.



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

Right now the Greeting() function takes in a name and displays Hi, my name is to that person.

You've learned about functions before but there are a few differences with composable functions.



```kotlin
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
```

- You add the @Composable annotation before the function.
- @Composable function names are capitalized.
- @Composable functions can't return anything.

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.

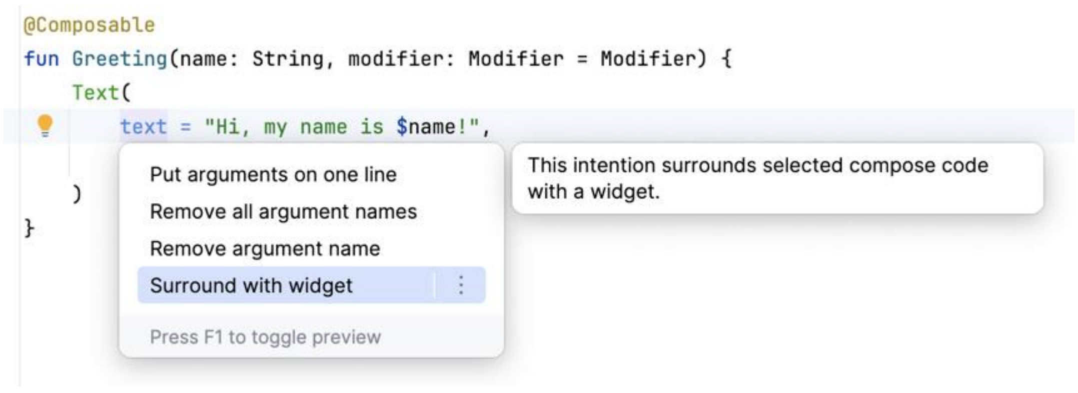
As you can see, the @Preview annotation takes in a parameter called showBackground. If showBackground is set to true, it will add a background to your composable preview.

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

5. 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.

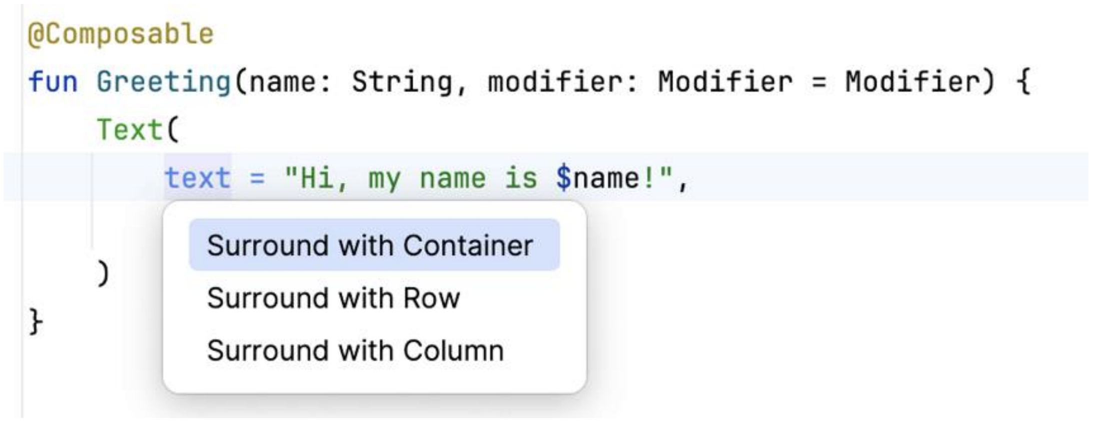
I. To surround the text with a Surface, highlight the line of text, press (Alt+Enter for Windows or Option+Enter on Mac), and then select Surround with widget.



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

    )
}
```

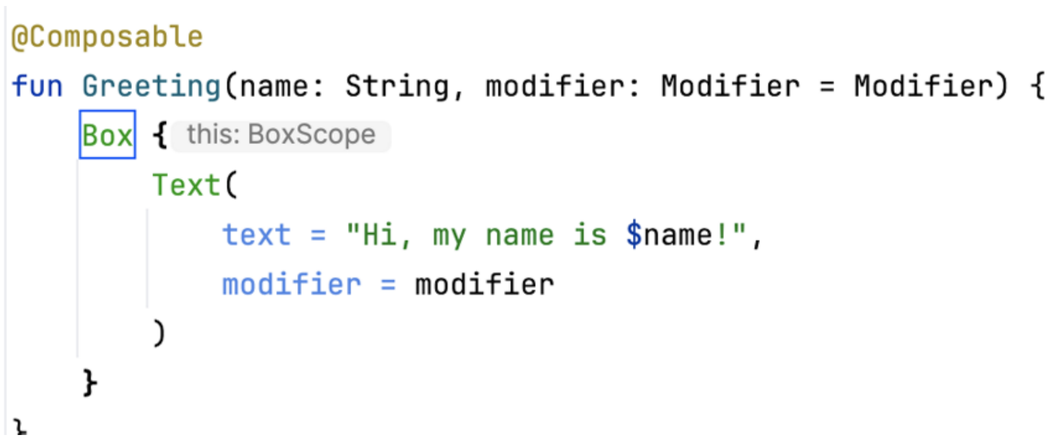
Put arguments on one line
Remove all argument names
Remove argument name
Surround with widget
Press F1 to toggle preview
This intention surrounds selected compose code with a widget.

II. Choose Surround with Container.



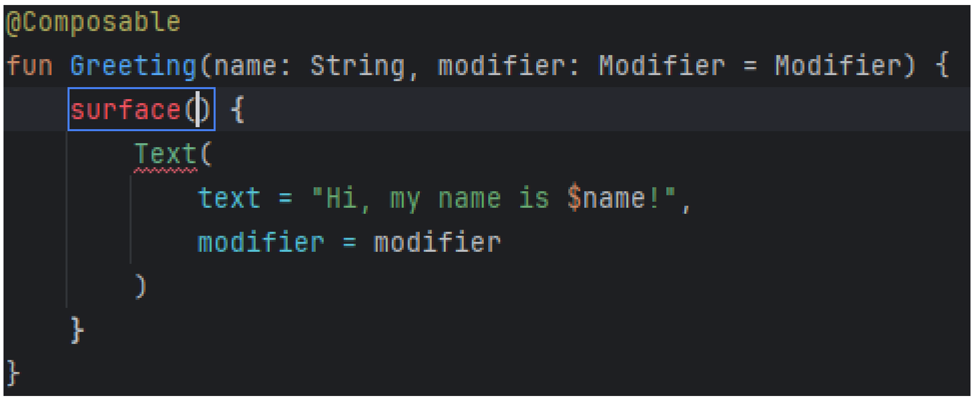
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hi, my name is $name!",

    )
}
```

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.



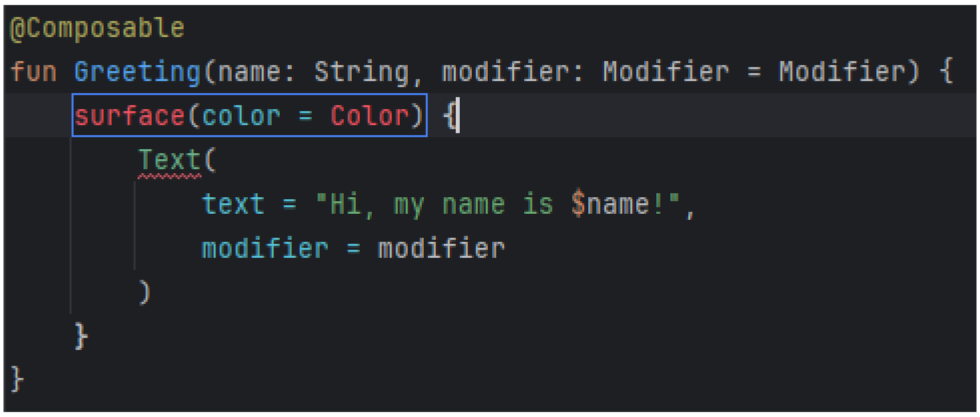
```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Box { this: BoxScope
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier
        )
    }
}
```

III. Delete Box and type Surface() instead.



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

IV. 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
        )
    }
}
```

V. When you type Color you may notice that it is red, which means Android Studio is not able to resolve this. To solve this add this statement;

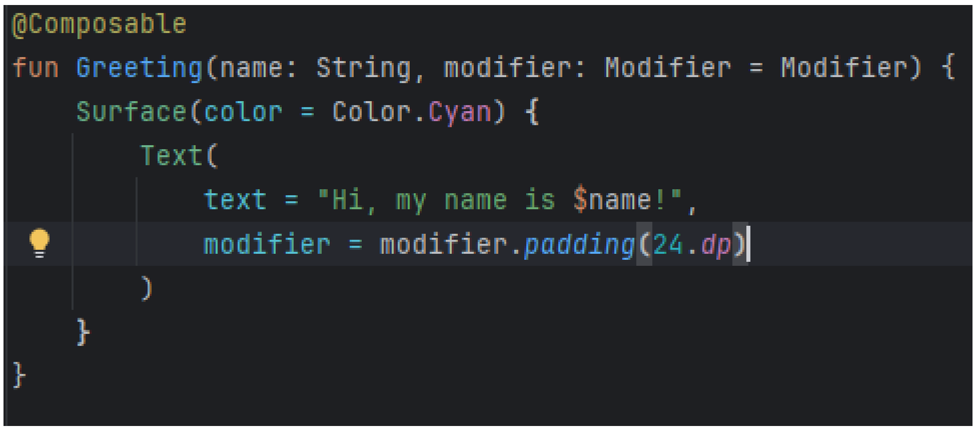
import androidx.compose.ui.graphics.Color

6. Add padding

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.

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

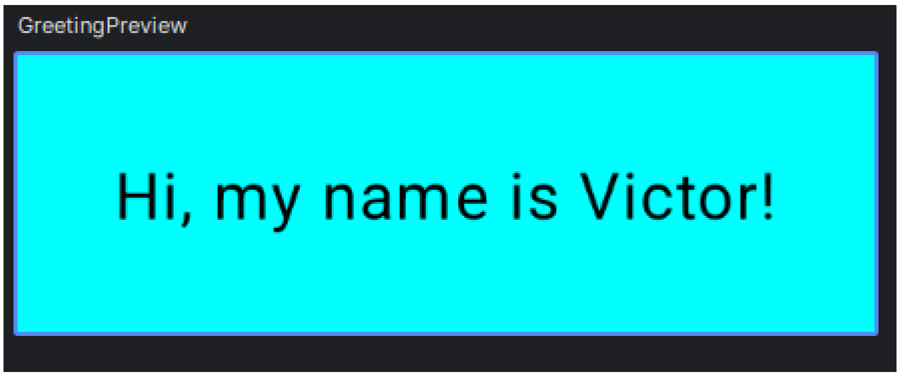


```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Surface(color = Color.Cyan) {
        Text(
            text = "Hi, my name is $name!",
            modifier = modifier.padding(24.dp)
        )
    }
}
```

II. Add these imports to the import statement section.

import androidx.compose.ui.unit.dp

import androidx.compose.foundation.layout.padding



7. Review the solution code

```
package com.example.maina

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Scaffold
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.maina.ui.theme.MainaTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MainaTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Greeting(
                            name = "Android",
                            modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}

@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() {
    MainaTheme {
        Greeting("Victor")
    }
}
```

8. Conclusion

You learned about Android Studio and built your first Android app with Compose.

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.