TP03: Build an app with text composables

1. Before we begin

In this **PRACTICE**[i], we use Jetpack Compose to build a simple Android app that displays a simple message on the screen:

```
Welcome,
I’m a Mobile Application.
From Android!
```

2. What we'll learn

- How to write composable functions, such as **`Text`**, **`Column`** and **`Row`** composable functions.
- How to display text in your app.
- How to format the text, such as changing text size (i.e., font size).

3. Create an Empty Compose Activity project

1) In the **Welcome to Android Studio** dialog, select **New Project**.
2) In the **New Project** dialog, select **Empty Compose Activity** and then click **Next**.
3) Enter **`Welcome`** in the **Name** field and then select a minimum API level of 21 (Lollipop) in the **Minimum SDK** field and click **Finish**.
4) Wait for Android Studio to create the project files and build the project.
5) Click **Run ‘app'** (in preference use a physical device).

The app should look like this screenshot:

When you created this mobile app with the **Empty Compose Activity template**, Android Studio set up resources for a basic Android app, including a **Hello Android!** message on the screen.

- ⊡ Composable functions

To Notice the Composable functions in code, we should follow these steps:

1) In Android Studio, open the `MainActivity.kt` file.
2) Scroll to the `DefaultPreview()` function and delete it. Add a new composable function, `WelcomePreview()` to preview the `Greeting()` function, as follows. As a good practice, functions should always be named or renamed to describe their functionality.

```
@Composable
fun WelcomePreview() {
    WelcomeTheme {
        Greeting( name: "Android")
    }
}
```

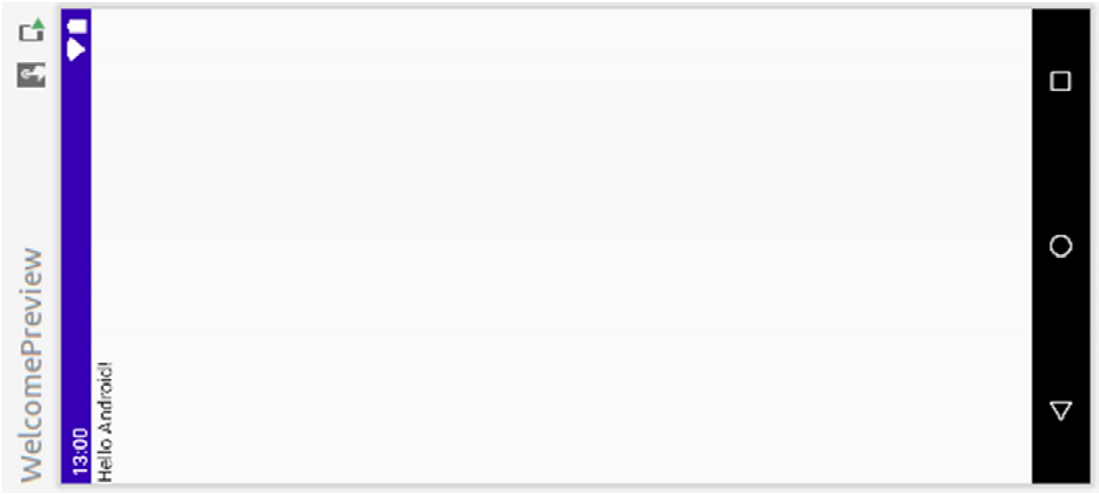
Composable functions can call other composable functions. The preview function is calling the `Greeting()` composable function.

Notice the previous function also has another annotation, a `@Preview` annotation, with a parameter before the `@Composable` annotation.

Annotations are means of attaching extra information to code. This information helps tools like the Jetpack Compose compiler, and other developers understand the app's code.

- ⊡ *Annotation with a preview title and the system UI (the phone screen)*-- multiple parameters

```
@Preview(showBackground=true,showSystemUi = true)
@Composable
fun WelcomePreview() {
    WelcomeTheme {
        Greeting( name: "Android")
    }
}
```



In the `WelcomePreview()` function, change the `"Android"` argument in the *`Greeting()`* function to your name.

```
@Preview(showSystemUi = true)
@Composable
fun WelcomePreview() {
    WelcomeTheme {
        Greeting( name: "Informatics")
    }
}
```

To view our new preview:

- ⊡ Build the code (The preview should automatically update).

Another way to update or refresh the preview is to click  **Build & Refresh** in the **Design** pane.

The code we added to the `WelcomePreview()` function with the `@Preview` annotation is only for previewing in the **Design** pane in Android Studio. These changes aren't reflected in the mobile app.

4. Change font size

We added text to our user interface, but it doesn't look like the final app yet. In this task, we learn how to change the size, text color, and other attributes that affect the appearance of the text element. We can also experiment with different font sizes and colors.

1) In the **MainActivity.kt** file, scroll to the Text() composable in the WelcomeGreeting() function.

2) Pass the Text() function a fontSize argument as a second named argument and set it to a value of 36.*sp*[1].

Android Studio highlights the .sp code because we need to import some classes or properties to compile our app.

3) Click **.sp**, which is highlighted by Android Studio.
4) Click **Import** in the popup to import the **androidx.compose.ui.unit.sp** to use the **.sp** extension property.

```
@Composable
fun WelcomeGreeting(message: String) {
    Text(text = "Hello $message!",fontSize=36.sp)
}
```

5. Add another text element

In our previous tasks, we added a welcome-greeting message to our friends. In this task, each student signs its message with his/her name.

1) In the **`MainActivity.kt`** file, scroll to the **`WelcomeGreeting()`** function.
2) Pass the function a **`from`** parameter of type String for your signature.

```
fun WelcomeGreeting(message: String,from: String) {
```

3) After the Welcome message **`Text composable`**, add another **`Text composable`** that accepts a text argument set to the from value and a **`fontSize`** named argument set to a value of **`24.sp`**.

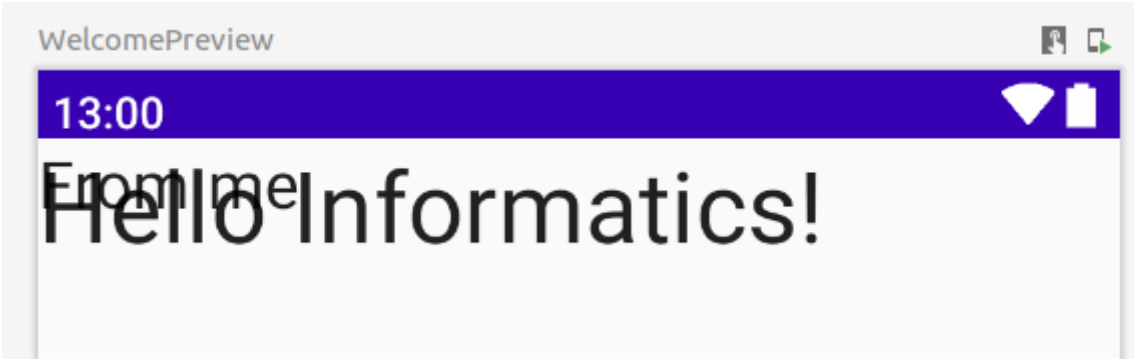
```
    Text(text = from,fontSize=24.sp)
```

4) Scroll to the **`WelcomePreview()`** function.
5) Add another String argument to sign your message, such as "- From me".

```
WelcomeGreeting( message: "Informatics", from: "-From me")
```

6) Click **Build & Refresh.**

[1] The *scalable pixels* (SP) is a unit of measure for the font size. UI elements in Android apps use two different units of measurement: *density-independent pixels* (DP), which you use later for the layout, and scalable pixels (SP). By default, the SP unit is the same size as the DP unit, but it resizes based on the user's preferred text size under phone settings.



A composable function might emit several UI elements. However, if you don't provide guidance on how to arrange them, Compose might arrange the elements in a way that you don't like. For example, the previous code generates two text elements that overlap each other because there's no guidance on how to arrange the two composables.

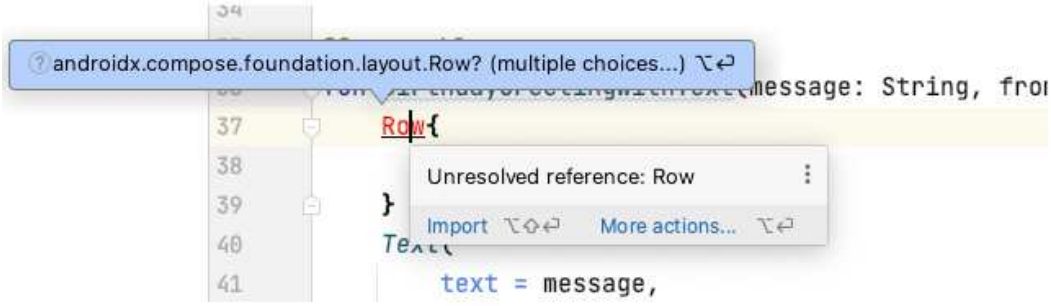
6. Arrange the text elements in a row

In this task, you arrange the text elements in your app in a row to avoid overlap.

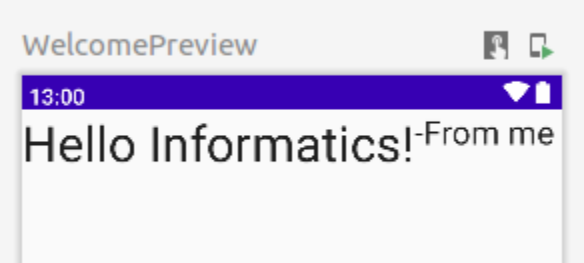
1) In the MainActivity.kt file, scroll to the WelcomeGreeting() function.
2) Add the Row composable around the text elements so that it shows a column with two text elements.

```
@Composable
fun WelcomeGreeting(message: String,from: String) {
    Row{ this: RowScope
    Text(text = "Hello $message!",fontSize=36.sp)
    Text(text = from,fontSize=24.sp)}
}
```

3) Click on Row, then Import and Select the package from the androidx.compose.ui class that begins with: `Row(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Argument.Horizontal, androidx....`



4) Click **Build & Refresh** to update the preview in the **Design** pane.

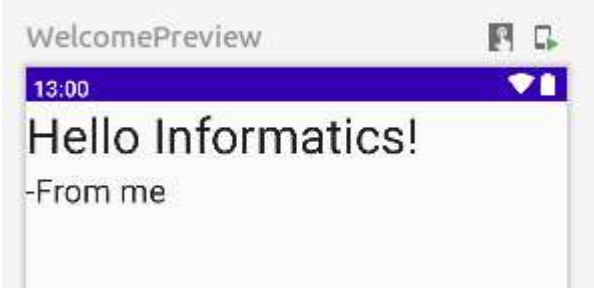


7. Arrange the text elements in a column

In this task, it is your turn to change the WelcomeGreeting() function to arrange the text elements in a column.

```
@Composable
fun WelcomeGreeting(message: String,from: String) {
    Column{ this: ColumnScope
    Text(text = "Hello $message!",fontSize=36.sp)
    Text(text = from,fontSize=24.sp)}
}
```

Import the column package when prompted by Android Studio.



8. Display the app on the device

Once you're happy with the preview, it's time to run your app on your device or emulator.

1) In the `MainActivity.kt` file, scroll to the `onCreate()` function.
2) Call the `WelcomeGreeting()` function from the `Surface` block.
3) Pass the `WelcomeGreeting()` function, your welcome greeting and signature.

```
WelcomeGreeting( message: "Informatics", from: "-From me")
```

4) Build and run your app using the emulator or a physical device.

[i] https://developer.android.com/codelabs/basic-android-kotlin-compose-text-composables