TP04: Add Resources to mobile App

1. Before we begin

In this **PRACTICE**, we learn how to add images and texts to our app.

2. What we'll learn

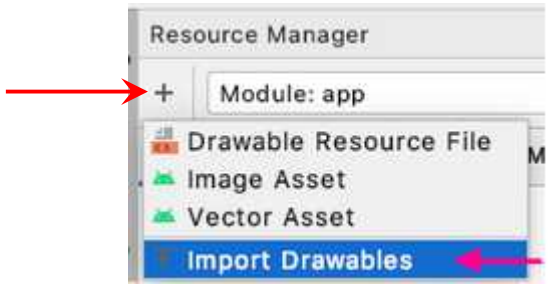
- ⊡ How to add an image or photo to our Android app.
- ⊡ How to display an image in our app with an **`Image`**[1] composable.
- ⊡ Best practices using **`Strings.xml`** `to insert texts` resource.

3. Upload an image to our project

In this task, we'll select an image and add it to our **`Welcome`** app.

a. In Android Studio, click **View > Tool Windows > Resource Manager** or click the **Resource Manager** tab next to the **Project** window.

b. Click **+** **(Add resources to the module) > Import Drawables**.



c. In the file browser, select the image file that you chose and then click **Open** (This action opens the **Import drawables** dialog).

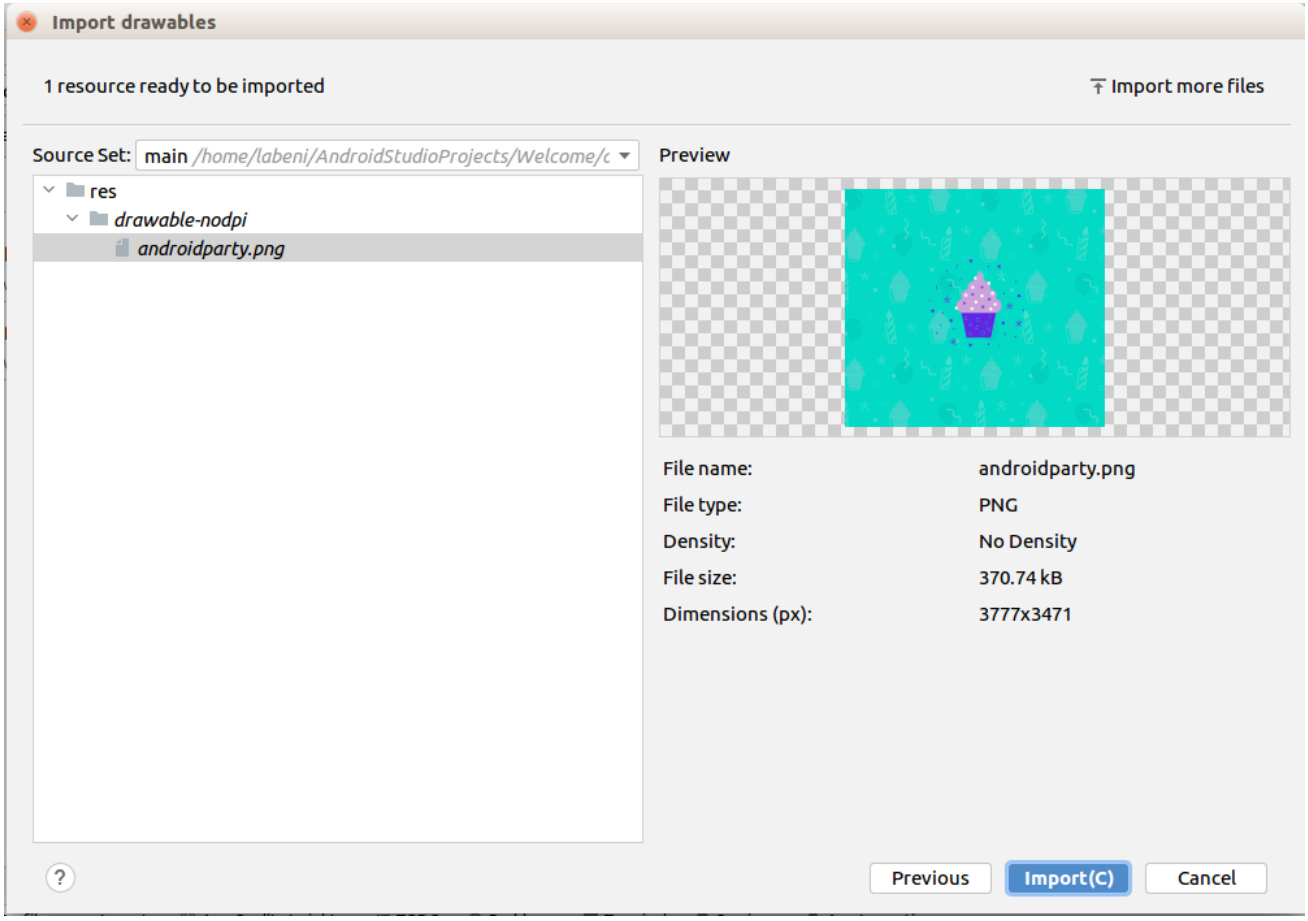
d. Android Studio shows you a preview of the image. Select **Density** from the **QUALIFIER TYPE** drop-down list. And then select **No Density** from the **VALUE** list and Click **Next**..

Android devices come in different screen sizes (phones, tablets, and TVs to name a few), and their screens also have different pixel sizes. That is, while one device has 160 pixels per square inch, another device fits 480 pixels in the same space. If you don't consider these variations in pixel density, the system might scale your images, which could result in blurry images, or large images that consume too much memory, or images that are sized improperly.

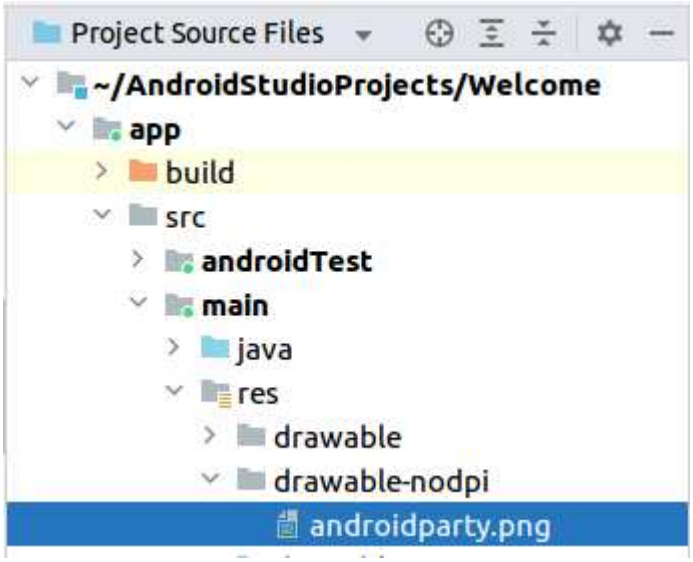
When we resize images that are larger than the Android system can handle, an out-of-memory error is thrown. For photographs and background images, such as the current image, the **androidparty.png**, you should place them in the **drawable-nodpi** folder, which stops the resizing behavior.

[1] https://developer.android.com/codelabs/basic-android-kotlin-compose-add-images

e. Android Studio shows the folder structure in which our image will be placed. Notice the **drawable-nodpi** folder. Click **Import.**



f. Click the Project Source Files tab. And then click **`app > src > main > res > drawable-nodpi`** to confirm that the image is uploaded successfully.



4. Add a composable function to insert an image

1. In the `MainActivity.kt` file, add a `WelcomeImg()` function before the `WelcomePreview()` function.

```
@Composable
fun WelcomeImg(message1: String, message2: String, from: String){
```

In the next task, we use the image, **`androidparty.png`** file, which you added in the previous task.

1. In the `WelcomeImg()` function, declare a `val` property and name it `image`.
2. Make a call to *`painterResource()`* function by passing in the *`androidparty`* resource. Assign the returned value to the `image` variable.

```
val image= painterResource(R.drawable.androidparty)
```

The `painterResource()` function loads a drawable image resource, and takes resource ID (`id= R.drawable.androidparty`) as an argument.

3. After the call to the `painterResource()` function, add an `Image` composable and then pass in the `image` as a named argument for the `painter`. add another named argument called `contentDescription` and set its value to `null`.

```
Image(painter=image,contentDescription = null)
```

Include the following import:

```
import androidx.compose.foundation.Image
```

4. Preview the Image composable

In this task, we preview the image composable and run the app on an emulator or device.

a) In the `WelcomePreview()` function, replace the WelcomeG`reeting()` function call with a WelcomeI`mg()` function call.

Your function should look like this code snippet:

```
@Composable
fun WelcomeImg(message1: String, message2: String, from: String){
    val image= painterResource(R.drawable.androidparty)

    Image(painter=image,contentDescription = null)


}
```

```
@Preview(showSystemUi = true)
@Composable
fun WelcomePreview() {
    WelcomeTheme {
        WelcomeImg( message1: "Welcome", message2: "I'm a Mobile Application", from: "-From me")
    }
}
```

b) In the **Design** pane, click  **Build & Refresh**.

Notice that you can't see the text anymore because the new function only has an `Image` composable, but not a `Text` composable.

5. Add Box layout

1. In the `WelcomeImg()` function, add a `Box` composable around the `Image` composable. At the end of the `Box` composable, call the `WelcomeGreeting()` function, and pass it the birthday message and the signature as shown:

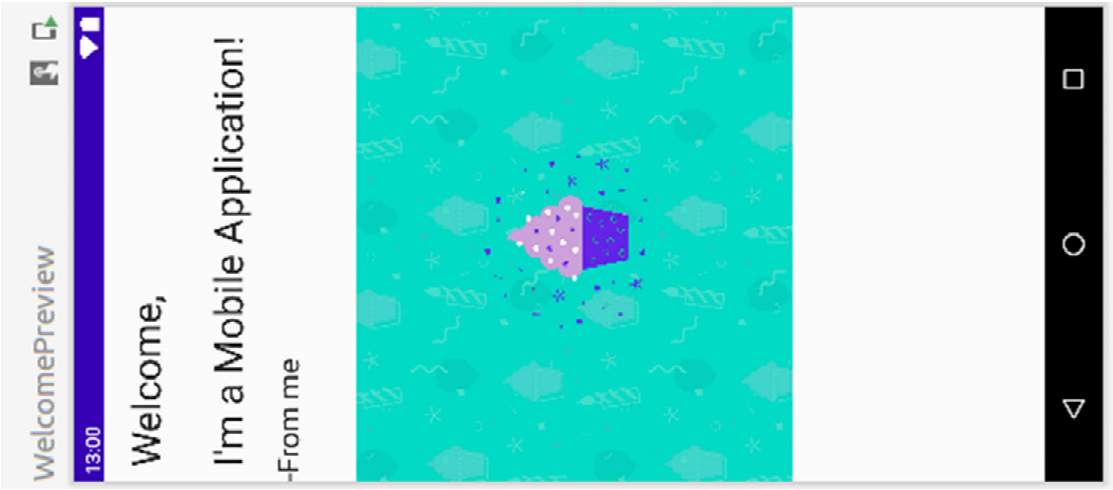
```
@Composable
fun WelcomeImg(message1: String, message2: String, from: String){
    val image= painterResource(R.drawable.androidparty)
    Box { this: BoxScope
    Image(painter=image,contentDescription = null)
        WelcomeGreeting(msg1 = message1, msg2 = message2, from =from )}

}
```

2. Click **Build & Refresh** in the **Design** pane to see the updated preview.
3. To make the above changes reflect in the emulator or a device, in the `onCreate()` function, replace the `WelcomeGreeting()` function call with the `WelcomeImg()` function call.
4. Run the app on an emulator or device.

Notice that the image is as wide as the app screen, but the image is anchored to the middle of the screen.

There's a lot of whitespace at the bottom of the screen that doesn't look very attractive. In our next task, we will fill the width and height of the screen, and scale the image to fill the entire screen.



6. Scale content

We have added the image to our app and positioned it. Now, we need to adjust the scale type of the image, which says how to size the image, to make it fullscreen.

We use the `ContentScale.Crop` parameter scaling, which scales the image uniformly to maintain the aspect ratio so that the width and height of the image are equal to, or larger than, the corresponding dimension of the screen.

Add a `ContentScale` named argument to the image.

```
Image(painter=image,contentDescription = null, contentScale = ContentScale.Crop)
```

import the `androidx.compose.ui.layout.ContentScale` interface when prompted by Android Studio.

- Click **Build & Refresh** in the **Design** pane.

The cupcake image should now fill the entire preview screen as we can see in this screenshot:

7. Text align, arrange and padding

In this task, we align, arrange and pad the text message to further beautify our app.

1. In the `MainActivity.kt` file, scroll to the `WelcomeGreeting()` function.
2. Add a `modifier` named argument to the `column` and then assign it a value of `Modifier.fillMaxSize()`. This will set the height and width of the column layout to the maximum available height and width.
3. Add `verticalArrangement` and `horizontalAlignment` properties to the column. Arrange the text elements to the top of the align them center horizontal.
4. To the welcome message text composable function add a `modifier` argument and set it to `Modifier.padding()`, and then pass it an argument set to `12.dp`.

```
@Composable
fun WelcomeGreeting(msg1:String,msg2:String,from: String){
    Column(modifier=Modifier.fillMaxSize(),
        verticalArrangement=Arrangement.Top,
        horizontalAlignment=Alignment.CenterHorizontally){ this: ColumnScope
        Text(text = "$msg1,",fontSize=33.sp,modifier = Modifier.padding(12.dp), color = Color.Blue)
        Text(text = "$msg2!",fontSize=33.sp,modifier = Modifier.padding(12.dp),color= Color.Yellow)
        Text(text = from,fontSize=24.sp)}
}
```

5. Run the app again, and then notice the paddings.

8. Display text from resource

Generally we use string resources instead of hardcoding `Text`[2] values, as we can share the same strings with our Android Views as well as preparing our app for internationalization:

```
Text(stringResource(R.string.hello_world))
```

[2] https://developer.android.com/jetpack/compose/text

For inserting a paragraph in our app, first we should declare a variable called "**paragraph**" with the corresponding text in **strings.xml** file like shown below:

MainActivity.kt | strings.xml

Edit translations for all locales in the translations editor.

```
<resources>
    <string name="app_name">Welcome</string>
    <string name="paragraph">Android Studio affiche plusieurs fenêtres utiles
        indiquées dans l'onglet tout en bas :\n
        -Logcat Affiche tous les messages émis par la tablette courante.\n
        - Messages: Les msgs du compilateur et du studio.\n
-Terminal Shell linux: permettant de lancer des commandes dans le dossier du projet.
    </string>
</resources>
```

Then we use the variable named **paragraph** in **WelcomeImg()** function to insert the corresponding **text** using **Text Composable** function as follows:

```
@Composable
fun WelcomeImg(message1: String, message2: String, from: String){
    val image= painterResource(R.drawable.androidparty)
    val text = stringResource(R.string.paragraph)

    Box { this: BoxScope
        Image(painter=image,contentDescription = null, contentScale = ContentScale.Crop)
        WelcomeGreeting(msg1 = message1, msg2 = message2, from =from )
        Text(text = text,fontSize=26.sp,
            modifier = Modifier.padding(top=500.dp,start=10.dp,end=10.dp),
            fontWeight = FontWeight.Bold,
            fontStyle= FontStyle.Italic ,
            textAlign = TextAlign.Justify ,
            fontFamily = FontFamily.Cursive)
        }
}
```

- To see the effect of this last click **Build & Refresh** in the **Design** pane.

- Or run it directly in a physical device like: