CS4131
Mobile Application Development

Name: ______________________________ (   ) Date: ______________

Chapter 0: Kotlin and Android Studio

0.1 Setting Up Android Studio

For this module, Android Studio will be the main code editor to develop and build your Android applications. While Java is the official language of Android development and is still supported by Android Studio, in May 2019, Kotlin was announced as the preferred language for Android application developers.

Hence, the purpose of this set of notes is to guide you through setting up of Android Studio and a basic introduction to Kotlin.

You may download the latest version of Android studio installer in the following website:

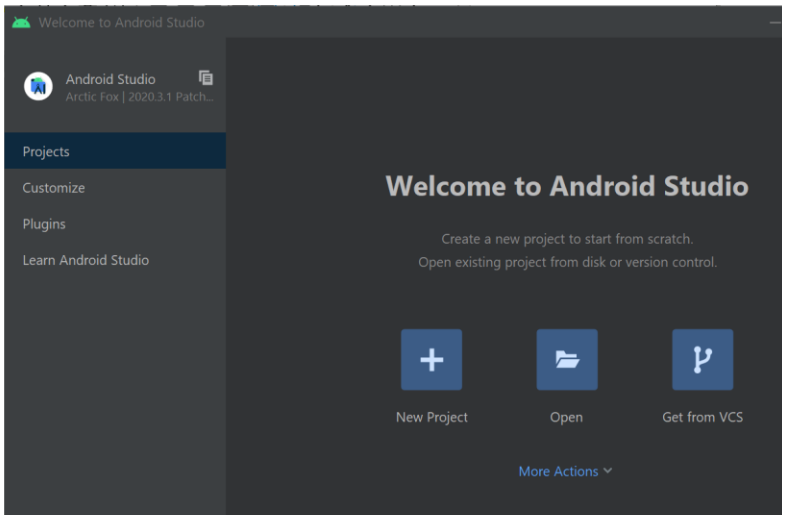
- https://developer.android.com/studio
- Ensure that the Android Studio version is at least 2020.3.1

Note that in this set of notes, it is assumed that you are familiar with how to create JavaFX programs within SceneBuilder as this set of notes will bring in elements of GUI design from JavaFX. Also, this set of notes will assume that you are familiar with the IntelliJ (Community Edition) interface for coding in Java.

After installing Android Studio (just let the installer install what has been pre-set for you),

- Run Android Studio and "Do not import settings" when prompted.
- After Android Studio opens, update any "Kotlin plugin updates", then restart the interface.
- Go through the environment setup wizard, choose "Standard" when prompted for setup, let Android Studio install any required SDKs, then "Finish.

You should see the following interface after the above is done:

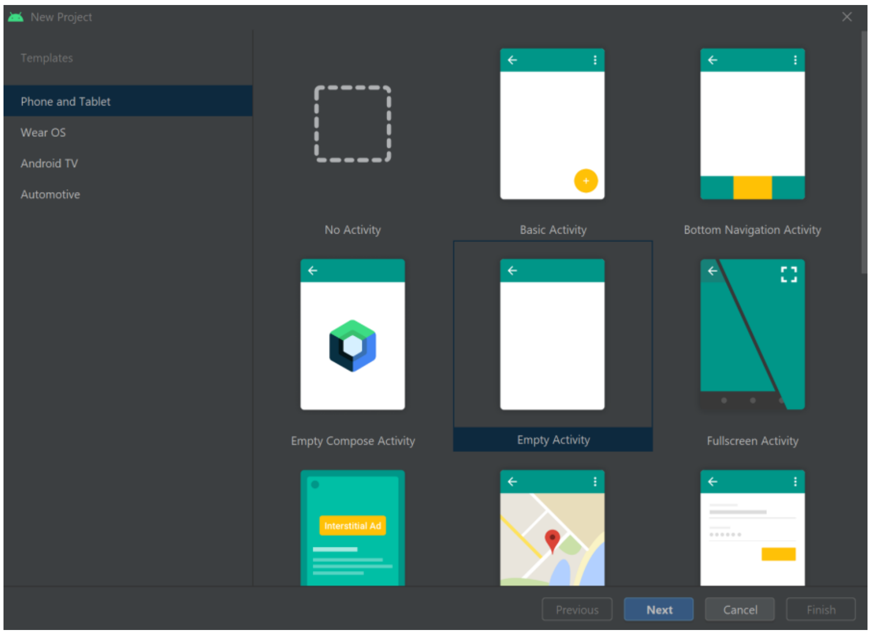


Welcome to Android Studio

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

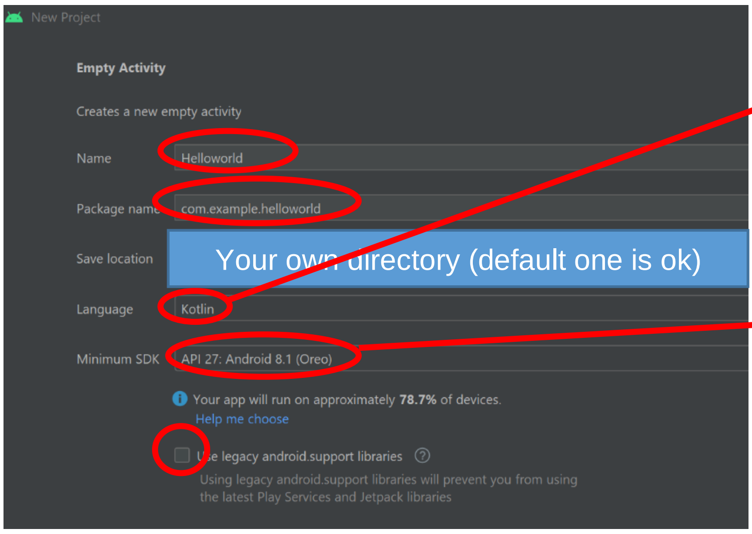
0.2 Setting Up a New Project

Select “New Project” in the Android Studio interface, and you will reach this screen:



In a nutshell, activities are like your various UI windows in your application.

Select “Empty Activity” and click “Next”. Ensure the following is entered / selected:


New Project

Empty Activity

Creates a new empty activity

Name: Helloworld

Package name: com.example.helloworld

Save location: Your own directory (default one is ok)

Language: Kotlin

Minimum SDK: API 27: Android 8.1 (Oreo)

Your app will run on approximately **78.7%** of devices.
Help me choose

☐ Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

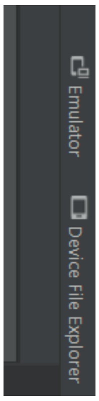
If you are going to code the app in Java, you will select “Java” in the drop-down list. For this set of notes, you will choose Kotlin.

The minimum Android SDK your app will target. Note that it cannot be too low (neither too high). Either will cause certain libraries required to not work properly.

You can finally click on finish after.

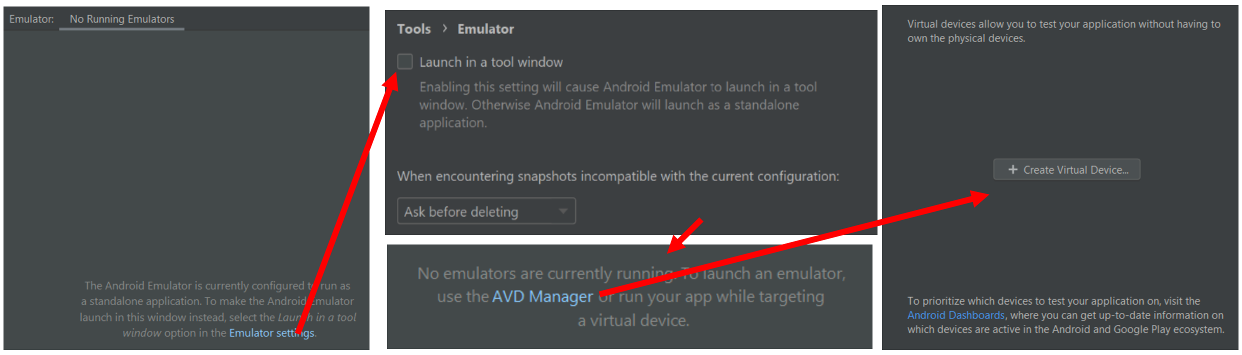
Note that the **<u>startup for will take quite a while</u>** on the first run (they are updating the Gradle plugins) When a “New Project” is created, you will see bottom right of your window, Android Studio is setting up the Gradle environment. For your information, Gradle is a build automation tool for multi-language software development which supports quite a significant number of common programming languages.

0.3 Setting Up the Emulator



For the module assessments, you will have to set up an Android emulator. Fortunately, Android Studio has an in-build Android Virtual Device (AVD) which you can use to run your application.

At the bottom-right side of Android Studio, you will see the following "Emulator" and "Device File Explorer" as seen in the diagram on the right. To set-up the AVD, click on the "Emulator" tab and you will see the link "Emulator settings". Click on the button and you will see the settings, ensure the checkbox "Launch in a tool window" is NOT selected (this will open the emulator in a separate window which will prevent certain problems from happening), then click on Apply. Close the Settings window and you will see a prompt to use the AVD Manager. Click on the AVD Manager link then click on "+ Create Virtual Device".



Emulator: No Running Emulators

The Android Emulator is currently configured to run as a standalone application. To make the Android Emulator launch in this window instead, select the *Launch in a tool window* option in the Emulator settings.

Tools › Emulator

☐ Launch in a tool window

Enabling this setting will cause Android Emulator to launch in a tool window. Otherwise Android Emulator will launch as a standalone application.

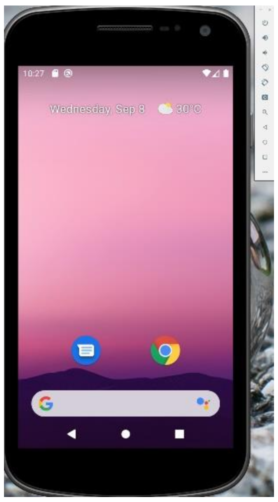
When encountering snapshots incompatible with the current configuration: Ask before deleting

No emulators are currently running. To launch an emulator, use the AVD Manager or run your app while targeting a virtual device.

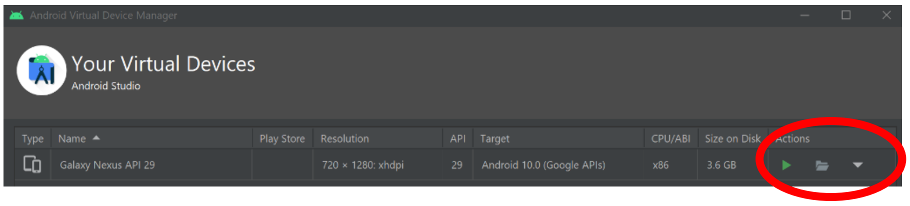
Virtual devices allow you to test your application without having to own the physical devices.

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.

Select **Galaxy Nexus** as your virtual device and set the system image to be **release Q, Android 10 (Google APIs)**. Selecting an image which has the (Google APIs) label with grant you access to various Google APIs which can facilitate development. Some images have the Google Play label which gives you Google Play support but is more restrictive as certain security features have to be implemented to your app which may impede app troubleshooting. **Ensure that you download release Q so that you can select it.**



After release Q is downloaded and you are able to click on the Next button, leave the settings as default (most importantly, leave it as portrait orientation), then click Finish. Click on the play button under the Actions column and the emulator will start-up and run:



Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name ▲	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Galaxy Nexus API 29		720 × 1280: xhdpi	29	Android 10.0 (Google APIs)	x86	3.6 GB	▶

Note that you should ensure your emulator is running BEFORE you build and run any app. While building your app, if Android studio stays stuck at "Waiting for target device to come online" message, stop the emulator completely, go to the AVD manager and select "Wipe Data", then restart the emulator.

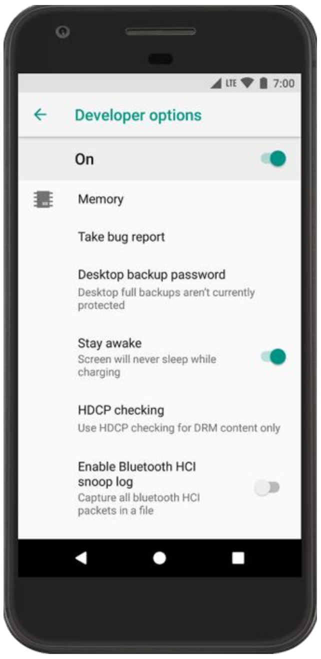
If you wish to go back to the AVD manager to change any settings for your emulator, go to Tools, then select AVD Manager. Alternatively, you can click on the icon on the top right corner of the Android Studio window.

0.4 Connecting your own Android phone via USB

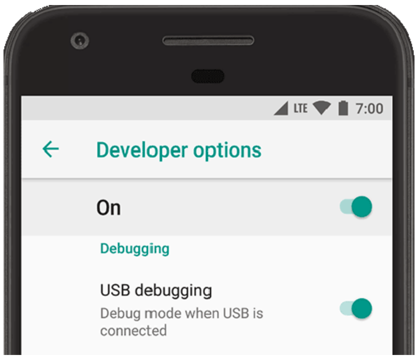
If you own an Android phone, you may alternatively wish to connect your own phone through a USB cable. Ensure the following options are chosen as you connect your phone to the computer (note that the options may differ in wording between devices):

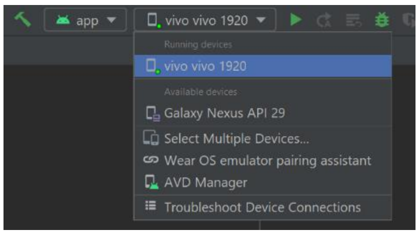
- Ensure that the USB option is set to “Files and photos” and NOT charging
- Ensure that the cable you are using is appropriate for file transfer. Note that some cables cannot perform any form of file transfer

After you connect your phone with Android Studio open, you will need to ensure that your phone is set to **USB Debugging** mode. To do that, you will need to first enable Developer Options. On the more recent OS (Android 9 and above), you can do a this by, on your device, going to Settings → About Phone → Build Number, then enabling developer options. Note that have devices have measures to ensure that the Developer Options is not accidently available to the user to prevent any accidental mishaps. In the more recent versions, instructions will be given to **tap on the Build Number seven times** for phones with Developer Options not enabled


Developer options
On
Memory
Take bug report
Desktop backup password
Desktop full backups aren't currently protected
Stay awake
Screen will never sleep while charging
HDCP checking
Use HDCP checking for DRM content only
Enable Bluetooth HCI snoop log
Capture all bluetooth HCI packets in a file

After the Developer Options are set, you will see that your Settings now have a Developer Options selection. Go into that selection and enable USB debugging.


Developer options
On
Debugging
USB debugging
Debug mode when USB is connected


app
vivo vivo 1920
Running devices
vivo vivo 1920
Available devices
Galaxy Nexus API 29
Select Multiple Devices...
Wear OS emulator pairing assistant
AVD Manager
Troubleshoot Device Connections

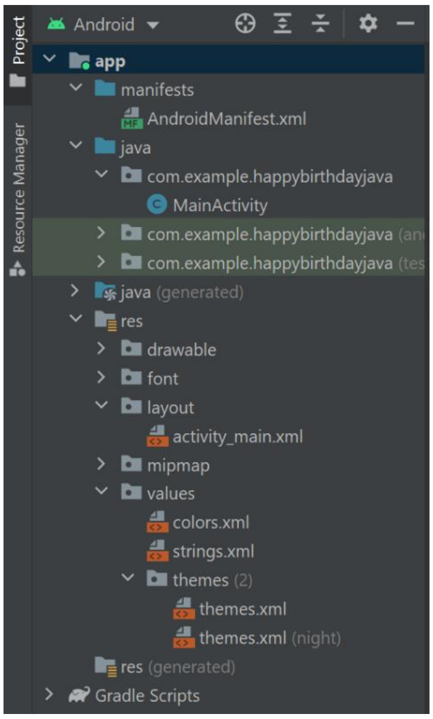
Once USB debugging is enabled, your Android device can communicate with your machine running Android Studio, through the Android Debug Bridge (ADB). Within Android Studio, you can now see that the phone you connected can now be selected to run your project build.

Note that when running your project in your device, the debugger will install the project directly into your device. After the installation, the app will be able to run standalone without the need to connect your device to Android Studio to run it, until you perform a rebuild for any updates. Also, if you wish to free up space in your phone, you will have to uninstall the apps.

Connecting your device to perform a test run of your projects will be important, for scenarios where your app uses any internal services like telephony, text messaging or the camera, or if your app involves interaction with the user through motion or environmental sensors. Testing for such scenarios will not work on the AVD emulator.

For more details on the on-device developer options, you may wish to view the following link: https://developer.android.com/studio/debug/dev-options

0.5 Application Project Folder



In the project folder, you will notice a project folder structure somewhat similar to the IntelliJ interface. While there are many files within the project app folder, there are a few main files you will have to concern yourself with for now:

1) AndroidManifest.xml

- This file is a compulsory file in your project folder which contains information about your app to the Android build tools, Android OS and Google Play
- Android Studio automatically updates the mainfest file when you introduce downloaded libraries or features.
- It is strongly recommended that you DO NOT edit the manifest file unless you are confident in customising the libraries, build and features of your app.

2) MainActivity.kt

- This file contains the “Controller” code which runs the “Main Activity” window of your app
- .kt is the extension for a Kotlin file. If your app runs using a Java code, it will be a .java file

3) activity_main.xml

- This file contains the code for the application interface of the “Main Activity” window itself. XML is the acronym for “Extensible Markup Language” and governs the layout, views and controls of the UI.

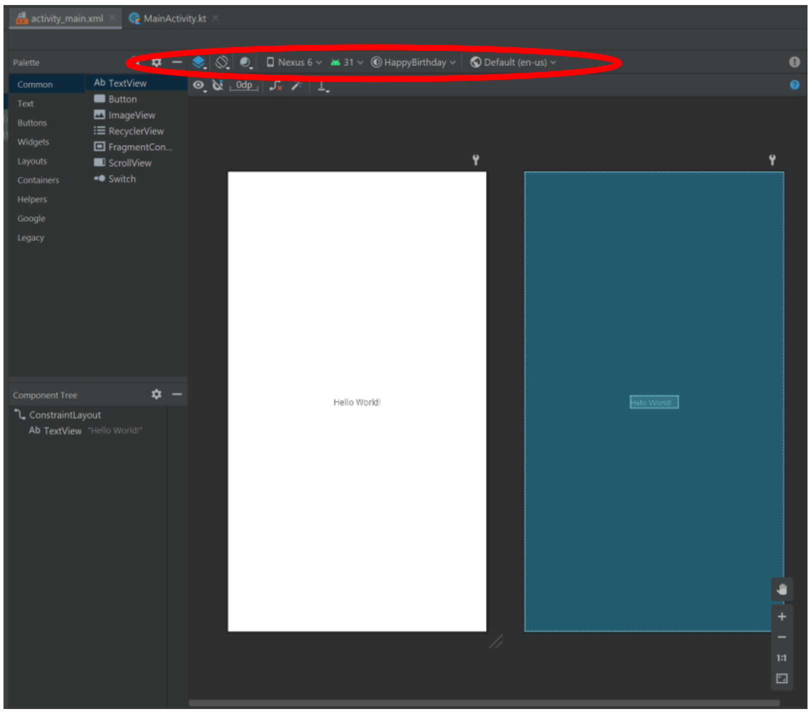
4) The res folder

- The resource folder contains multiple XML files which govern various aspects of the interface like the colours, fonts and more.
- You will notice folders like drawable, font, layout, mipmap and values.
- The layout will contain the XML files directly linked to your Activity windows. In this case, activity_main.xml is contained within the layout folder.
- The values folder, in essence, provides you with a way to define **meaningful names** to fonts, colours, strings and other components that you may **wish to use repeatedly in your application design.**
  - It is ***good app development practice*** to leave resources (fonts, colours, even strings) separate from the the main code and call upon them when needed based on your defined meaningful names.
- The themes folder is present to maintain a level of **consistency** of the different aspects of your UI when new Activities are created within the same application. Consistency in the colour and font is one of the key aspects in **enhancing the user experience of the application**.

In the next few sections, we will be exploring how to build a simple Hello World app with a working button which will be able to change the text in your application.

0.6 XML Editor Interface

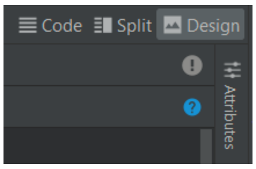
First, we want to deal with the interface itself. Click on the “activity_main.xml” tab. Note that XML is the acronym for Extensible Markup Language, and basically governs the layout, views, controls and aesthetics, similar to the function of the FXML file in a JavaFX application:

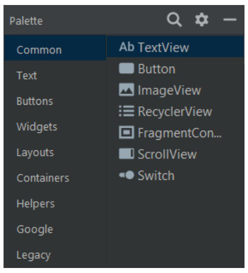


The part circled above contains the following:

- Toggle Design and Blueprint views.
    - Blueprint view provides a simplified outlined view of the UI, useful towards the final stages of development when the UI is more cluttered. Design view shows the actual design which will be seen on the phone.
- Toggle portrait / landscape / UI view customized for smart-watches, TV, etc
- Toggle day / night mode
- Nexus 6 Toggle the phone to view the UI design on a particular device model.

The right side of the XML editor toggles between XML code only, the design window only, or both. You may need one or the other throughout the process of designing your app.





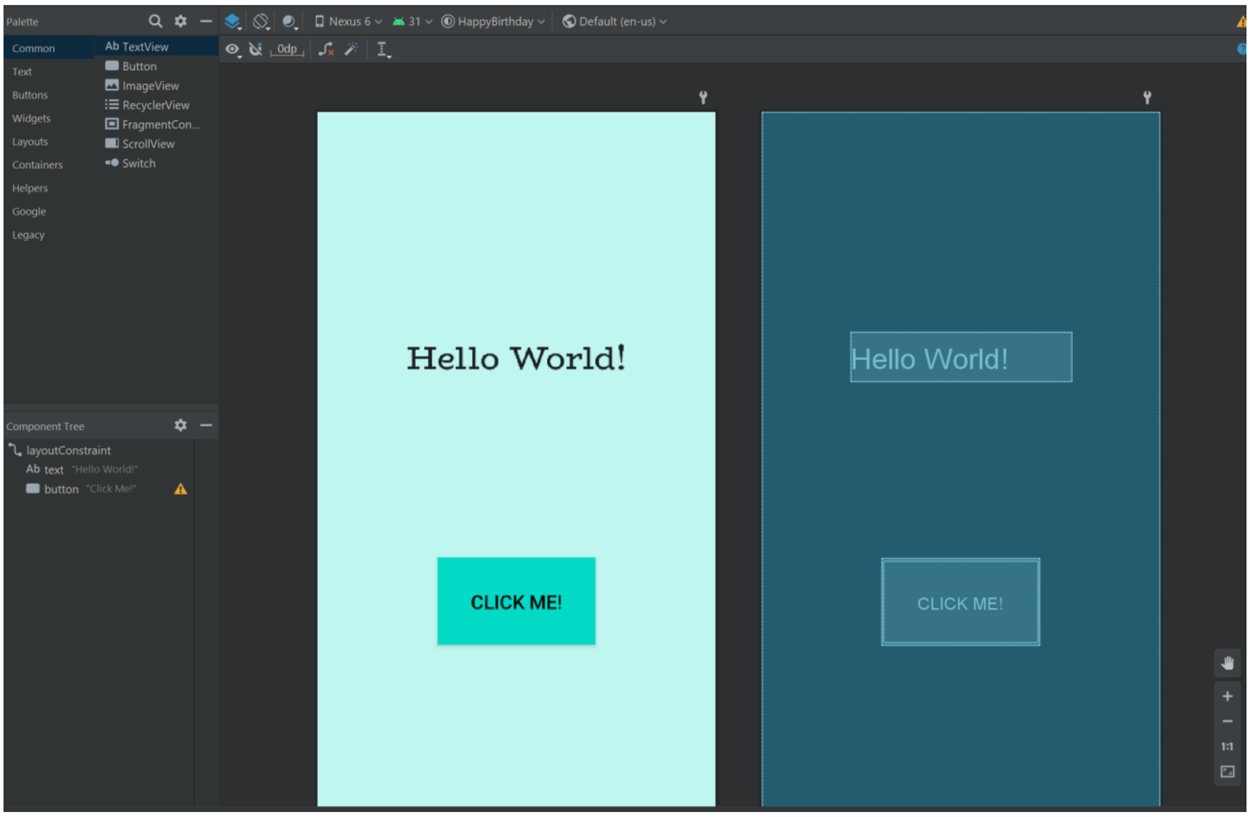
When you enter the Split or Design window, the left side of the XML editor has the “Palette” tab which opens to shows the list of controls, widgets and containers, similar to the SceneBuilder interface for JavaFX. To edit the attributes and properties of controls, widgets, etc, you can toggle the “Attributes” tab on the right side. It is strongly encouraged to explore the XML editor to get familiarised with it.

You may also go into https://www.w3schools.com/xml/ to learn about the features of XML.

0.7 Building the Hello World App Interface

For our Hello World app, we will explore the use of a TextView and a Button placed onto a Constraint layout. In the Palette, under the "Common" category, you will see the options for TextView and Button. you may "drag and drop" these controls into your main app preview window.

The outcome of this section will be to obtain the following:



For each control, **<u>after selecting</u>** the control, you may open the "Attributes" tab on the right. You will first notice an "id" field. This id field allows you to define a name for your control, which as you may recall, is similar in functionality to JavaFX's "fx:id" on SceneBuilder. You can also scroll down to the "**All Attributes**" to define some of its attributes. We shall define the following:

Control	Attributes
TextView	id: text fontFamily: @font/biorhyme text: Hello World! textAppearance: @style/TextAppearance.AppCompat.Body1 textSize: 34sp typeface: normal
Button	id: button background: Change to any background and observe any changes style: @style/Widget.AppCompat.Button text: Click Me! textSize: 20sp typeface: normal

You will notice that as you define the attributes, these same attributes will now be in a new section at the top called “Declared Attributes” for easy access to edit later. To edit the colour of the “background”, click anywhere on the “background” and you will be able to edit the attributes (including the id) of the Constraint Layout, to which you will then be able to edit the “background” attribute.

As you define the attributes, you will observe the following:

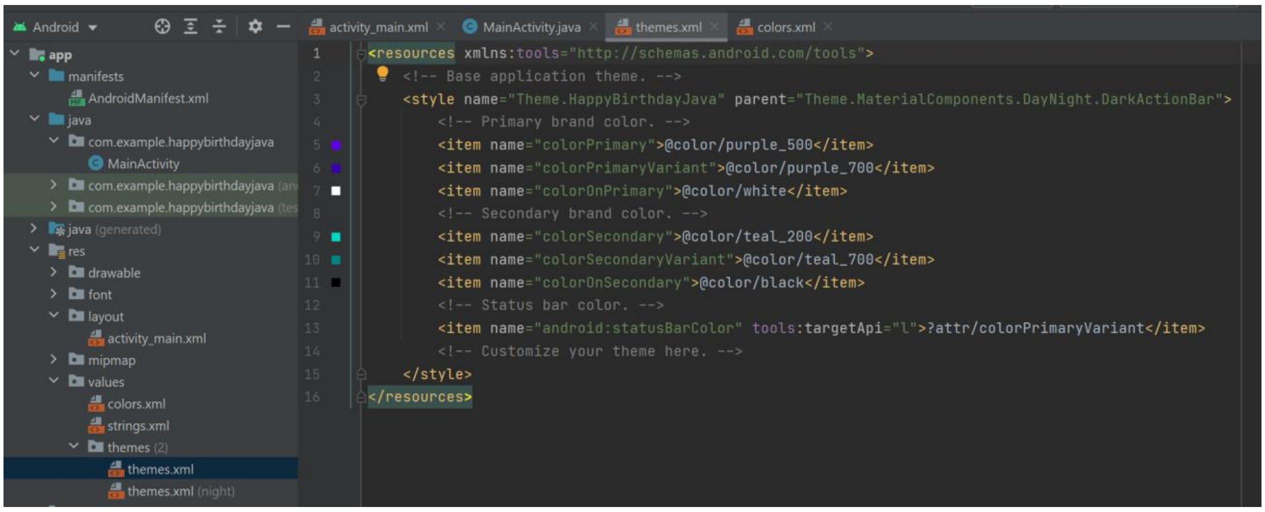
1) You are unable to change the background colour of the button (it will stay purple)
2) Using the fontFamily dropdown list for the TextView, you will not be able to find the “biorhyme” font. If you copy-paste what is written in the previous page into the field, you will notice a prompt for a font download.

The above issues will be addressed in the following (and more):

<u>Defining / Redefining Resources</u>

Android Studio maintains a resource folder with multiple XML files which define not only layout and view of the entire app in general, but also maintains a ***strong consistency of elements*** such as colour, fonts and even strings throughout the entire app, <u>**which is vital in the user experience of an app**</u>.

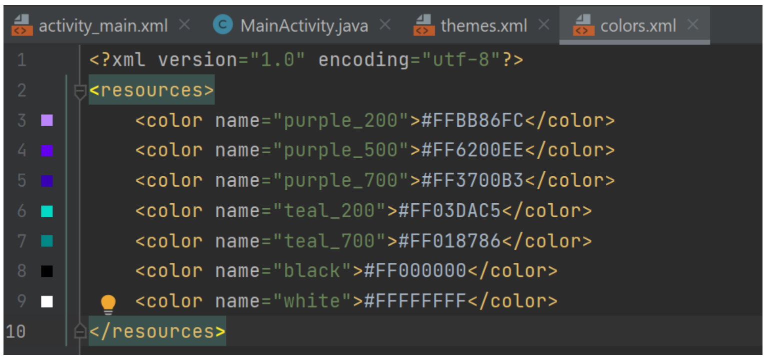
To resolve the first problem of changing the background colour of the button, while maintaining the consistent theme defined for the entire app (the “right way”), inside the res folder, go to the themes.xml and you will see the following:



```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.HappyBirthdayJava" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

You will now notice that the primary colours are all purple, which explains why the buttons are purple and you are unable to edit the colours! You may also notice that there is a file named “themes.xml (night)”. This enables you to also define a “night-mode” theme for your app as well!

Before changing the colour of you application, go to “colors.xml” and you will notce that there are colour names defined with a corresponding hexadecimal code:

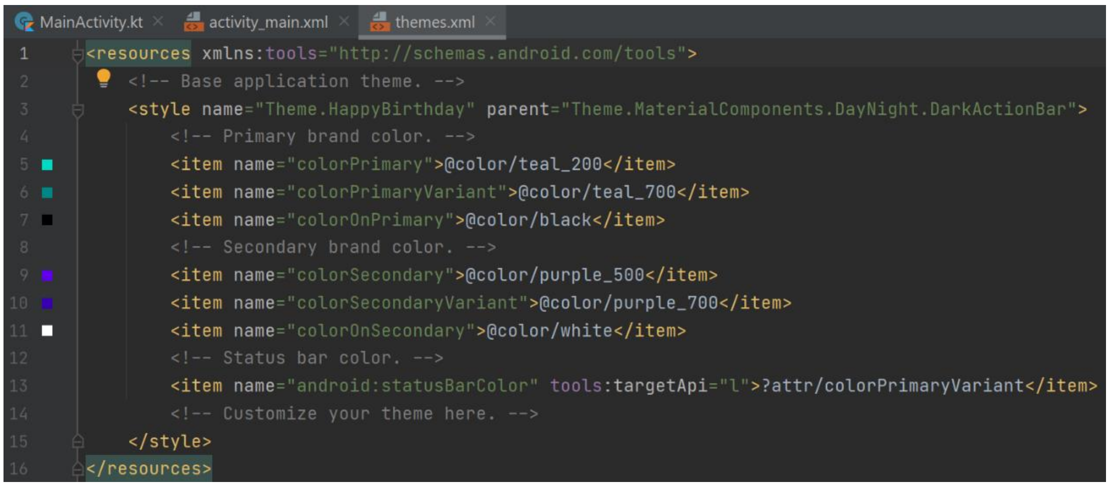


activity_main.xml × MainActivity.java × themes.xml × colors.xml ×

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

This “colors.xml” file serves as a **mapping** to map each hexadecimal colour code to a “color name”, and thereafter, you will be able to reference that hexadecimal colour code, through that name in all your other XML files as well as main code!

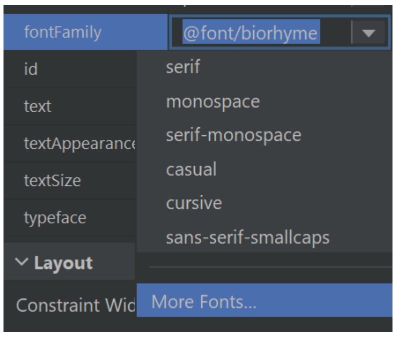
Hence, to change the colours of your buttons, what you can do is to simply switch the colours of the primary and secondary colours, as seen below, or you can be creative and define your own colours in the “colors.xml” file and use those defined colours in your theme.



MainActivity.kt × activity_main.xml × themes.xml ×

```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.HappyBirthday" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/teal_200</item>
        <item name="colorPrimaryVariant">@color/teal_700</item>
        <item name="colorOnPrimary">@color/black</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/purple_500</item>
        <item name="colorSecondaryVariant">@color/purple_700</item>
        <item name="colorOnSecondary">@color/white</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Downloadable Resources

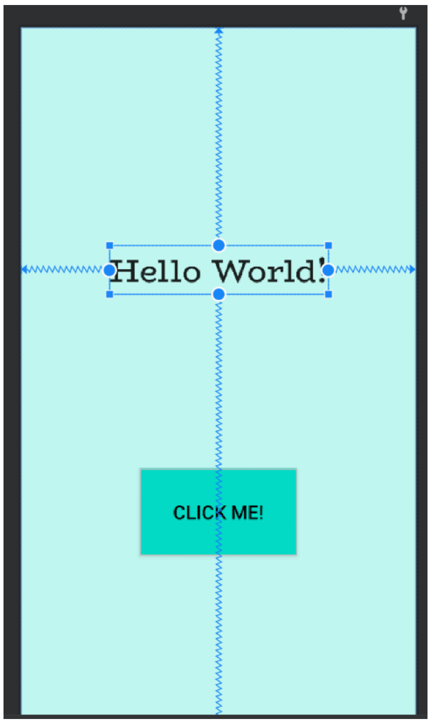


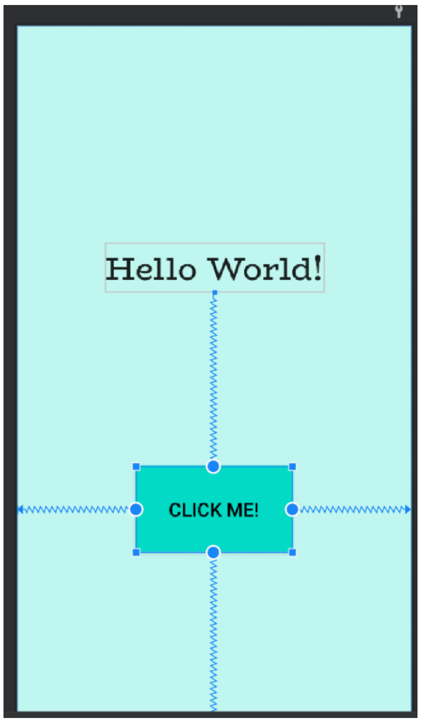
Android Studio allows you to download resources which may be readily available on their online repository. For the case of the fontFamily issue, you can scroll all the way to the bottom of the fontFamily drop-down list to see “More Fonts”. Selecting that option will open a new window which will allow you to browse through various fonts for download. Downloaded fonts used will be added to the res folder.

Such a feature is important when introducing elements of **typography**, an aesthetic and functional requirement to enhance the user experience of the app.

Defining constraints in Constraint Layout

While adjusting placement of the different controls on the design preview window, you may notice some circular nodes on each edge of the control. These nodes define the constraints the control will have with respect to the activity window or other controls. You can "click and drag" each of the circular nodes to the edges of the design window or other controls, to form a zig-zag line which connects itself to the edges of the design window:





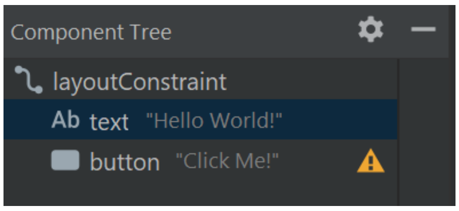
As you can see on the left, the "Hello World!" TextView is **constrained** to the edges of the design window and the button is constrained to the left, right and bottom edges of the window and the TextView.

These constraints allow the developer to have good control over the relative positions of all their components on a flat 2D hierarchy, allowing easy creation of complex and dynamic views.

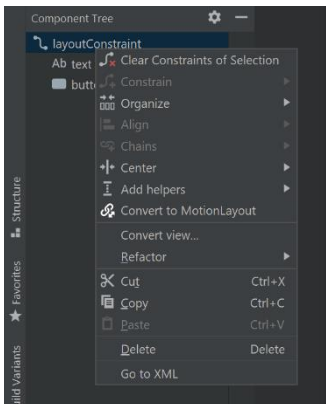
The constriant layout will be further explored in the later chapters of this module.

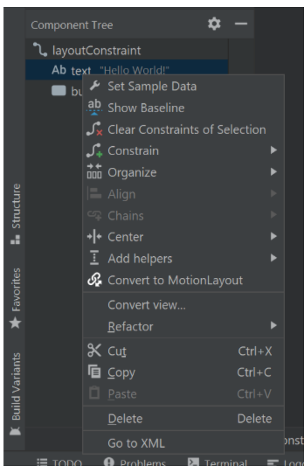
Component Tree

Similar to SceneBuilder for JavaFX has a Scene Hierarchy or a Scene Graph, you may see the hierarchy of the layout and controls in the component tree. The component tree is located in the bottom left side of the XML editor (directly at the bottom of the Palette tab). This allows you to visualise the relative layers of the layout and control elements, which will become important the moment the app design becomes sufficiently complex.



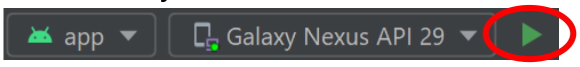
Right-clicking on either a layout or control in the component tree will provide you with addiitonal options which may give you good control and aid you when your app design becomes quite complex:



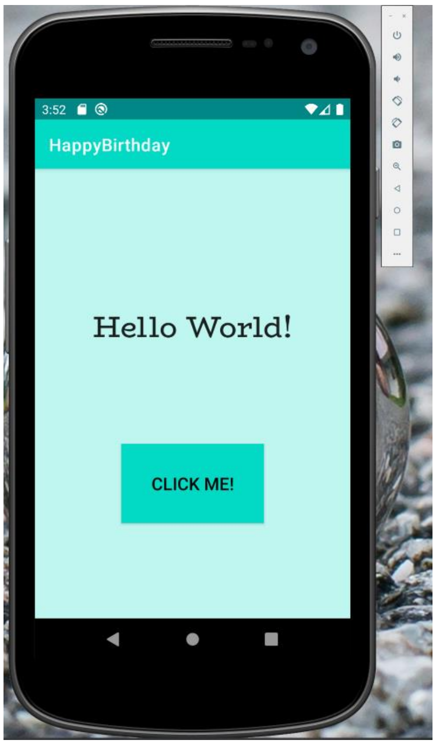


It is strongly recommended that you try and experiment with the options shown after the right-click.

0.8 Building and Running the Interface on the Emulator

After the emulator is running (go back to section 0.3 if you have already forgotten how to start the emulator), at the top right corner of Android Studio, ensure that your emulator is selected and click on the Play button: 

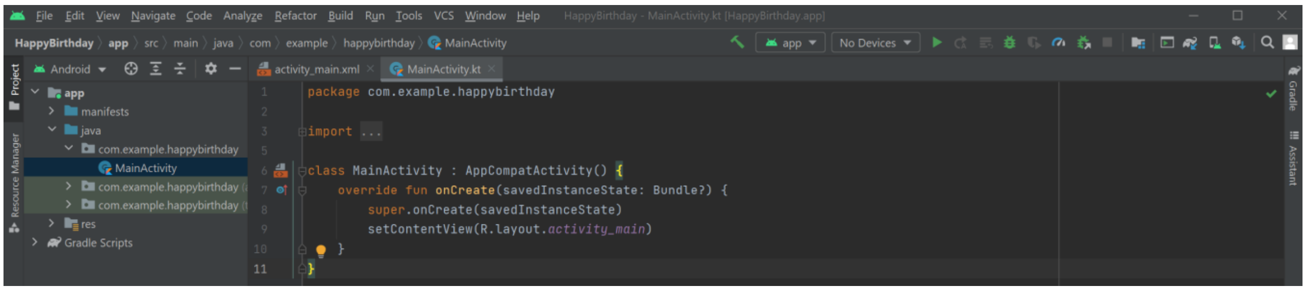
This will build the entire project and you will be able to see your app running in your emulator. You will notice that your app currently has no functionality; clicking on the button does nothing in the app. Hence, we now want to code functionalities on the app.



The main language that will be covered for this set of notes will be Kotlin, as Kotlin is now the preferred language for Android app development. Kotlin is an object-oriented language which is similar to Python but its syntax has an almost one-to-one correspondence to the Java syntax, which makes it easy to pick up for any programmer who is familiar enough with Java.

0.9 Coding App Functionalities in Kotlin

You will notice that the coding interface looks very similar to IntelliJ. It is indeed similar, as in fact, you can install an Android Studio plugin on IntelliJ and still code Android apps on the platform.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HappyBirthday - MainActivity.kt [HappyBirthday.app]
HappyBirthday > app > src > main > java > com > example > happybirthday > MainActivity
app
No Devices
Android
activity_main.xml
MainActivity.kt
app
manifests
java
com.example.happybirthday
MainActivity
com.example.happybirthday
com.example.happybirthday
res
Gradle Scripts

```
package com.example.happybirthday

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

As seen in the screenshot above, select the “MainActivity.kt” file. This file will act as the “Controller” code for the corresponding activity window you have developed in the previous section.

As mentioned in the previous section, Kotlin is now the preferred language to develop Android apps and has an almost one-to-one similarity to Java syntax wise. Hence, the code for “MainActivity.kt” to get the app functionalities working will be shown side by side with its equivalent Java code.

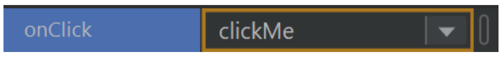
Note that the Kotlin code is to be copy-pasted BELOW the package declaration:

<u>Kotlin Code</u>:

```kotlin
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.TextView

class MainActivity : AppCompatActivity() {
    private var text : TextView? = null
    private var button : Button? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        text = findViewById<TextView>(R.id.text)
        button = findViewById<Button>(R.id.button)
    }

    // Called when the user taps the "Click Me!" button
    fun clickMe(view: View) {
        // Do something in response to button
        text?.setText(R.string.output_str)
    }
}
```

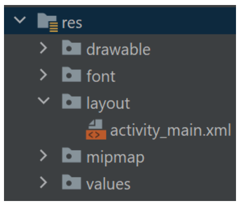
<u>Java Equivalent Code</u>:

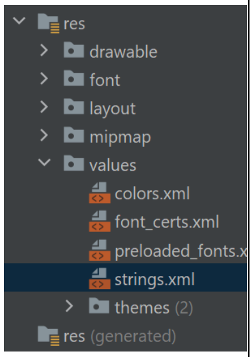
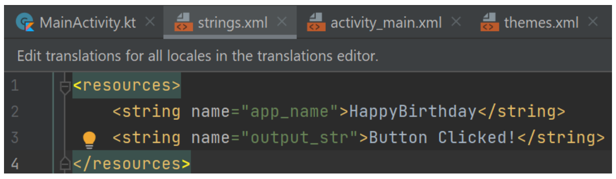
```java
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private Button button = null;
    private TextView text = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = findViewById(R.id.button);
        text = findViewById(R.id.text);
    }

    // Called when the user taps the "Click Me!" button
    public void clickMe(View view){
        // Do something in response to button
        text.setText(R.string.output_str);
    }
}
```

After implementing the code on the previous page, you will need to link the Button to the corresponding method which it will run upon tapping. Hence, you will need to go back to the XML editor, go into the Design Window, select the Button and go to All Attributes section, then ensure that onClick is set to the clickMe method: 
onClick clickMe

To explain the code, we will do a direct comparison between the Kotlin code and its equivalent Java code:

Lines	**Explanation**
Kotlin line 1 to 5 Java line 1 to 5	The libraries which have to be imported for the app. • `androidx.appcompat.app.AppCompatActivity` – this is the required base class which will allow you to use newer features on older Android devices. • `android.os.Bundle`– this is the required class which supports the transfer of data between activities. • The other imports are to support the TextView, Button and the required parameter signature for the clickMe method. Note that the syntax for importing is the almost same for Java and Kotlin (Kotlin has no need for semicolons)
Kotlin line 7 Java line 7	The Activity class, which acts like the "Controller" for the app project needs to extend from the AppCompatActivity class (recall inheritance from OOP) Note the syntax for class declaration and to perform inheritance in Kotlin. Note that Kotlin's **default visibility modifier is public**.
Kotlin line 8 to 9 Java line 8 to 9	Kotlin only has two primitive data types, var and val. • var is known as a **<u>mutable variable</u>**, and variables will only hold the var data type if they are to be edited sometime later in the code • val is known as an **<u>immutable variable</u>**, and is equivalent to a final variable in Java, where they cannot be edited. • In Kotlin, there is no need to declare specific data types like int, char, strings etc. (notice that this feature is similar to Python) As class attributes are private visibility, the private keyword is needed. Notice the way attributes are declared and initialized. In the Kotlin syntax, the ? character is required to tell the compiler that the variable has a possibility of being null, and is normally applied to objects during initialization or method calls which involve modification of the object.
Kotlin line 11 to 17 Java line 11 to 18	Activities are initialized through the onCreate() method. Note that in Kotlin, if a method override occurs, it needs to be declared. Also note that methods are all declared as functions with the keyword fun. • `super.onCreate(savedInstanceState)` – this is a required method call which will assure that your code runs alongside any pre-existing code in the onCreate superclass.

Lines	Explanation
	• `setContentView(R.layout.activity_main)` – this links the relevant XML file to the activity. The R class contains the definitions for all resources within the res folder of your project. Hence, `R.layout.activity_main` refers to the activity_main.xml file within the layout folder of the res folder of your project.  res drawable font layout activity_main.xml mipmap values • Similarly, to initialize the button and text variables, you have to use the `findViewById(R.id.<id in XML file>)` to link the variable to the object with the respective id within the activity_main.xml file.

Kotlin line 20 to 23
Java line 21 to 24
	Every method in Kotlin is declared as fun (short for function), the method name, then the parameters (note how the parameters are defined)
	
	When the setText() method is called, note that the ? character is used because text (the TextView object), has a possibility of it being a null. Also note that the "output_str" is defined in strings.xml in the values folder.  res
	drawable
	font
	layout
	mipmap
	values
	colors.xml
	font_certs.xml
	preloaded_fonts.x
	strings.xml
	themes (2)
	res (generated)  MainActivity.kt
	strings.xml
	activity_main.xml
	themes.xml
	Edit translations for all locales in the translations editor.
	`<resources>`
	`    <string name="app_name">HappyBirthday</string>`
	`    <string name="output_str">Button Clicked!</string>`
	`</resources>`

Instead of implementing a method which will run upon button tap, you can add a listener which will execute the change in TextView string the moment the "Click Me!" button is tapped:

Kotlin Code:

```
button?.setOnClickListener {
    text?.setText(R.string.output_str)
}
```

Java Equivalent Code:

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        text.setText(R.string.output_str);
    }
});
```

The table below will show some other useful Kotlin and Java syntax comparison:

Intent	**Kotlin**	**Java**

Defining functions	`fun main(){`
	`    // Some code`
	`}`
		`public static void main (String[] args){`
		`    // Some code`
		`}`

Printing	`print("Hello world")`
	`println("Hello world")`
		`System.out.print("Hello world");`
		`System.out.println("Hello world");`

Functions with return	`fun sum(a: Int, b: Int): Int {`
	`    return a + b`
	`}`
		`public static int sum (int a, int b){`
		`    return a + b;`
		`}`

Functions without return	`fun printSum(a: Int, b: Int) {`
	`    print("$a + $b = ${a + b}")`
	`}`
		`public static void printSum (int a, int b){`
		`   System.out.printf("%d + %d = %d", a, b, a+b);`
		`}`

If-else statement	`fun maxOf(a: Int, b: Int): Int{`
	`    if (a > b) {`
	`        return a`
	`    } else {`
	`        return b`
	`    }`
	`}`
		`public static int maxOf (int a, int b){`
		`    if (a > b)`
		`        return a;`
		`    else`
		`        return b;`
		`}`

For loop	`for (x in 1..10 step 2) {`
	`    print(x)`
	`}`
		`for (int x = 1; x <= 10; x += 2)`
		`    System.out.print(x);`

For-each loop	`val items = listOf("a","b","c")`
	`for (item in items) {`
	`    println(item)`
	`}`
		`String[] items = {"a","b","c"};`
		`for (String item : items){`
		`    System.out.println(item);`
		`}`

For loop	`val items = listOf("a","b","c")`
	`for (i in items.indices) {`
	`    println(items[i])`
	`}`
		`String[] items = {"a","b","c"};`
		`for (int i = 0; i < 3; i++){`
		`    System.out.println(items[i]);`
		`}`

While loop	`val items = listOf("a","b","c")`
	`var i = 0`
	`while (i < items.size) {`
	`    println(items[i])`
	`    index++`
	`}`
		`String[] items = {"a","b","c"};`
		`int i = 0;`
		`while (i < items.length){`
		`    System.out.println(items[i]);`
		`    i++;`
		`}`

Switch statement
 	
(When statement in Kotlin)
	`when (x) {`
	`    1 -> print("x is 1")`
	`    2 -> print("x is 2")`
	`    else -> {`
	`        print("x is not 1 or 2")`
	`    }`
	`}`
		`switch(x){`
		`    case 1:`
		`        System.out.print("x is 1");`
		`        break;`
		`    case 2:`
		`        System.out.print("x is 2");`
		`        break;`
		`    default:`
		`        System.out.print("x is not 1 or 2");`
		`}`

Try-Catch-Finally statement	`try {`
	`    // some code`
	`} catch (e: IOException) {`
	`    // handler`
	`} finally {`
	`    // optional finally block`
	`}`
		`try {`
		`    // some code`
		`} catch (IOException e) {`
		`    // handler`
		`} finally {`
		`    // optional finally block`
		`}`

You may explore more in the following website: https://kotlinlang.org/docs/home.html

For Kotlin code specific to Android app development: https://developer.android.com/kotlin/first

Note that for this module, it is NOT compulsory to code in Kotlin and you may choose to code your activities in Java. However, the module notes will only feature Kotlin codes to keep up to date with the trend of Kotlin now being the main language to be used for Android app development.

You can try coding in Kotlin in the following online website:

- https://developer.android.com/training/kotlinplayground

0.10 Logging The Development Process

During the development process, it will be useful to log the build process to debug any errors that may arise. To perform logging, you have to import the Log class and introduce a private constant String containing a tag to the activity name:

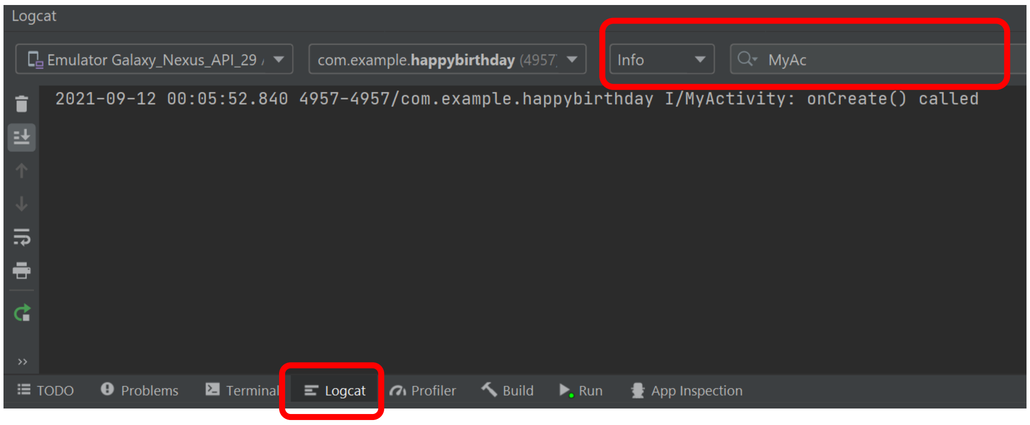
```
import android.util.Log
...
private val TAG : String = "MyActivity"
```

Within each method of the Activity Life Cycle (covered in Chapter 1), such as the onCreate() method, you may include the following code:

```
if (BuildConfig.DEBUG)
    Log.i(TAG, "onCreate() called")
```

Notice that the Log class contains the method "i". This means that the message `"onCreate() called"` is logged at the info level. Using other method names will determine which level the given message is logged under. The methods are, d (debug), i (info), v (verbose), w (warning), e (error), and wtf (what a terrible failure).

To view the log, at the bottom, you may click on the Logcat tab and select “Info” in the dropdown list. You will probably have to search for your tag to obtain your customised log message in the large haystack of log messages as you run your app.



```
2021-09-12 00:05:52.840 4957-4957/com.example.happybirthday I/MyActivity: onCreate() called
```

You may view more information regarding logging here:
https://developer.android.com/tools/debugging/debugging-log.html
https://developer.android.com/studio/debug/am-logcat.html

0.11 Building the APK

An APK, short for Android Package, is a file format used by the Android OS for distribution and installation of Android mobile apps. To build an APK, go to Build → Build Bundle(s) / APK(s) → Build APK(s). Android Studio will start building the APK in the background and a notification will appear. Clicking on "locate" will bring you to the physical directory the APK is located.

Otherwise, you can go to the physical directory where your project is located, then go to:
app → build → outputs → apk → debug. The APK is then located in the debug folder as the file app-debug.apk. Note that the APK, in practice, is only used for testing and not for actual deployment. To install and test this APK on an actual Android device, you will need to enable "Allow Unknown Sources" in your device settings when prompted upon opening your APK file.

To deploy a newly created app, a signed APK will need to be generated. What this means is that the APK needs to be digitally signed with a certificate whereby an upload key and keystore will need to be generated for the app before it can be uploaded into Google Play. For more details, you may read the following link: https://developer.android.com/studio/publish/app-signing

For this module, you will only need to build APKs just meant for testing, as highlighted in the first two paragraphs of this section.

0.12 Exploring More into Android Studio and Practice

You can now proceed on to doing Lab 0 to get some hands-on practice in developing apps with Android Studio. You are strongly encouraged to look at the references to detailed guides as to how Android Studio works and also to explore the more advanced features which may or may not be covered in the module.

[References]

[1] Android Developer Website: https://developer.android.com/
[2] Kotlin Language Documentation: https://kotlinlang.org/docs/home.html
[2] Android Logging: https://kotlinlang.org/docs/home.html