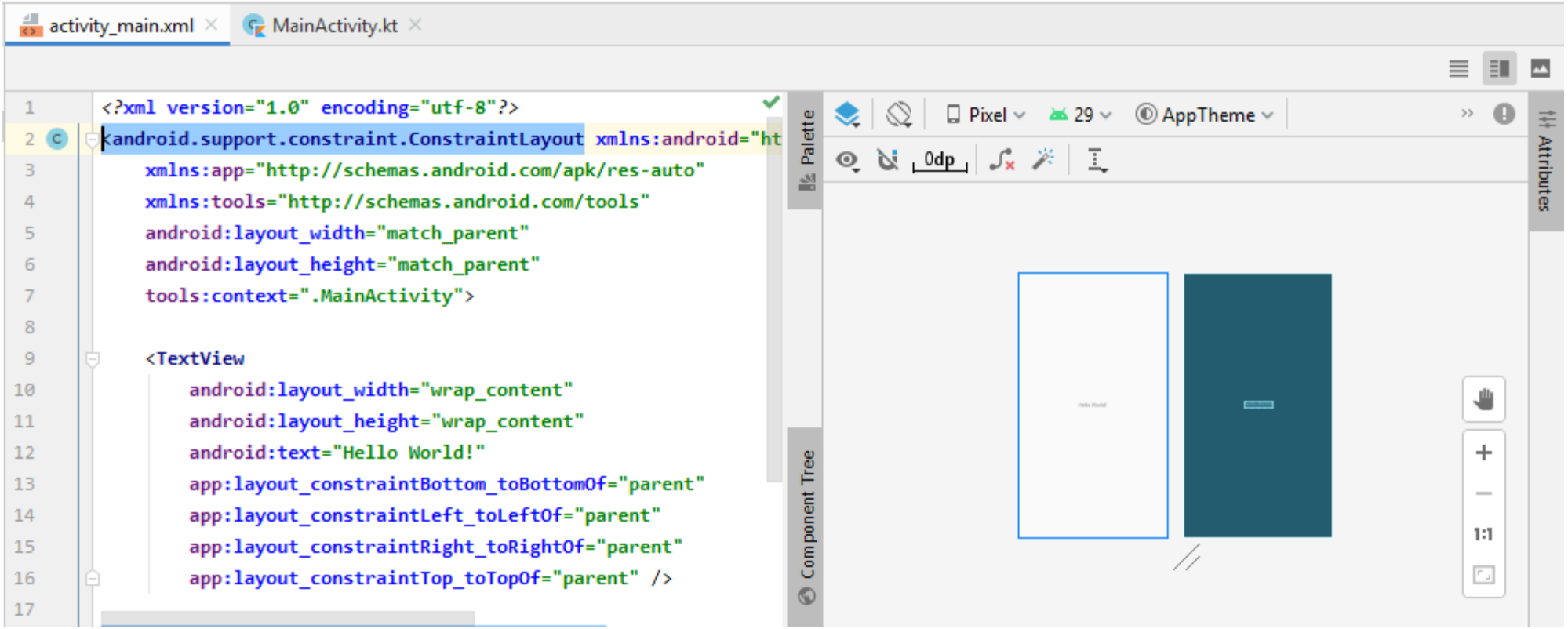
LECTURE 13: AN INTRODUCTION TO – THE POWERFUL MARKUP LANGUAGE

BY LINA HAMMAD & AHMAD BARGHASH

This Lecture, which will serve as your introduction to XML for Android development.

A BASIC INTRODUCTION TO XML AND MARKUP LANGUAGES

- In Android Studio, an XML file is created for each kotlin file you use in your application.



- You can create viewable elements and manage their attributes through XML

A BASIC INTRODUCTION TO XML AND MARKUP LANGUAGES

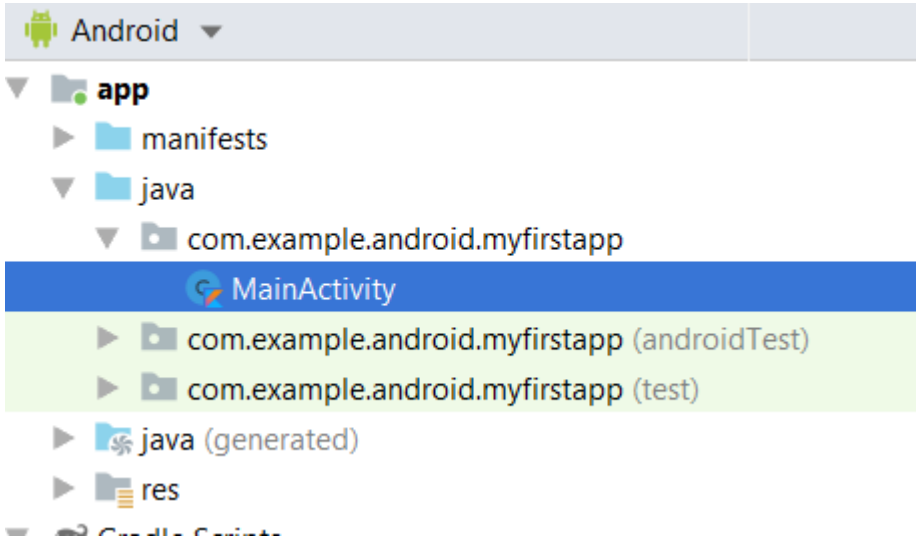
- XML stands for Extensible Markup Language, which gives us a clue to what it does.

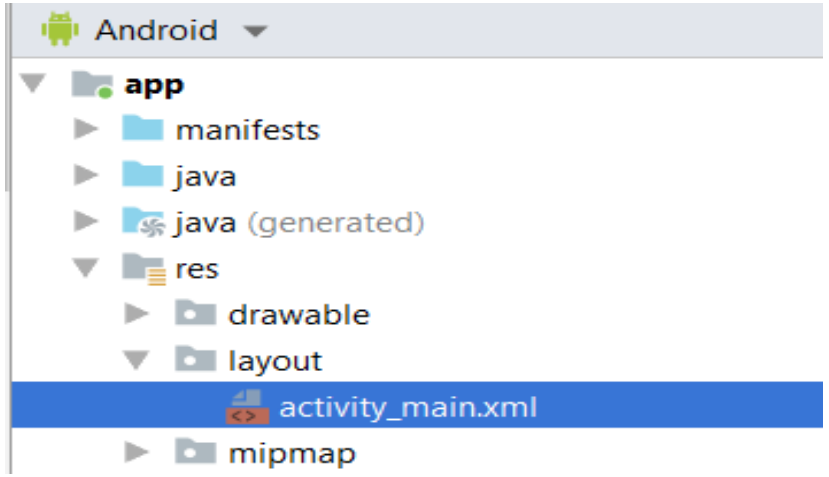
- A markup language is slightly different from a programming language. Whereas a programming language (C#, C++, Java, Kotlin, Python, BASIC) will allow you to define behaviors, interactions, and conditions; a **markup language is used more to describe data, and in this case, layouts**. Programming languages create dynamic interactions, whereas markup languages generally handle things like static user interfaces.

- In fact, XML was originally introduced by the World Wide Web Consortium (W3C) to deal with the inherent limitations of HTML. Specifically, HTML is not terribly readable for a computer, because it doesn't explain what anything on the page actually **is**.

XML IN ANDROID DEVELOPMENT

- When you create a new project in Android Studio, you will be greeted by a hierarchy of different files and folders, which can be a little daunting for complete beginners. It's a rather requiring an introduction to XML, no doubt!
- You just need to concentrate on two files for now: **MainActivity.kt** and **activity_main.xml**.
- To make life just a little simpler, Android Studio normally opens both these files as soon as it boots up.





XML IN ANDROID DEVELOPMENT

- You'll also notice that both these files have a little bit of code already in them. This is called "boilerplate code," A code that have to be included in many places with little or no alteration.
- One line in MainActivity.kt reads:

```
setContentView(R.layout.activity_main)
```

- This means the view of this kotlin code is controlled by the file **activity_main.xml** file,
- You can assign any XML file to any kotlin file.
- However, by default **MainActivity.kt** will always be the class (kotlin file) Android loads first when running your programs.

USING XML IN YOUR ANDROID APP

- XML describes the views in your activities (activity_main.xml), and Kotlin tells them how to behave (MainActivity.kt). To make changes to the layout of your app then, you have two main options.
- The first is to use the **Design view.** Open up the **activity_main.xml** file in Android Studio and get your first introduction to XML. You'll notice there are two tabs at the bottom (Or Up) of that window: **Design** and **Text**. The Text view will show you the actual XML code, but the Design view will let you manually edit the layout by dragging and dropping elements into the render of your activity.
- The second choice is to use the XML files. XML files can also help store strings. Using the Design view is easier for beginners, though it can lead to complications.

MANUALLY CREATING AN XML LAYOUT

- The structure of an XML layout file is actually quite straightforward and follows the hierarchical approach of the view tree. The first line of an XML resource file should ideally include the following standard declaration:

```
<?xml version="1.0" encoding="utf-8"?>
```

- This declaration should be followed by the root element of the layout, typically a container view such as a layout manager. This is represented by both **opening and closing tags** and any properties that need to be set on the view. The following XML, for example, declares a ConstraintLayout view as the root element, assigns the ID activity_main and sets match_parent attributes such that it fills all the available space of the device display:

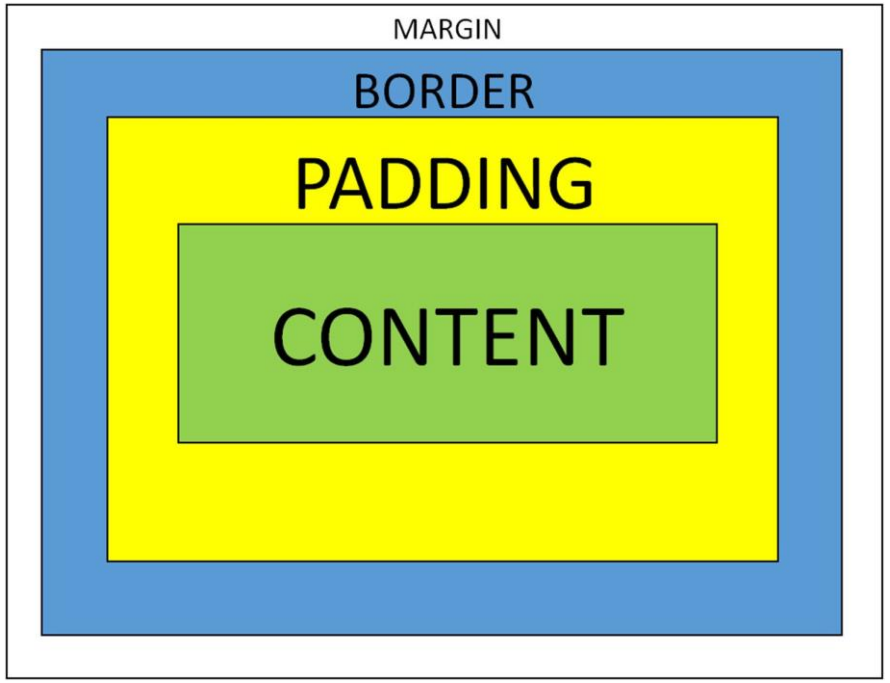
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

The slash denotes the end of a tag

ANDROID: LAYOUT_HEIGHT AND LAYOUT_WIDTH

- android:layout_height: specifies the basic height of the view.
- android:layout_width: specifies the basic width of the view.
- The base LayoutParams class just describes how big the view wants to be for both width and height. For each dimension, it can specify one of:
  1. MATCH_PARENT (previously FILL_PARENT), which means that the view wants to be as big as its parent (minus padding).
  2. WRAP_CONTENT, which means that the view wants to be just big enough to enclose its content (plus padding).
  3. An exact number: May be a dimension value, which is a floating-point number appended with a unit such as "14.5sp". Available units are: px (pixels), dp (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), and mm (millimeters).

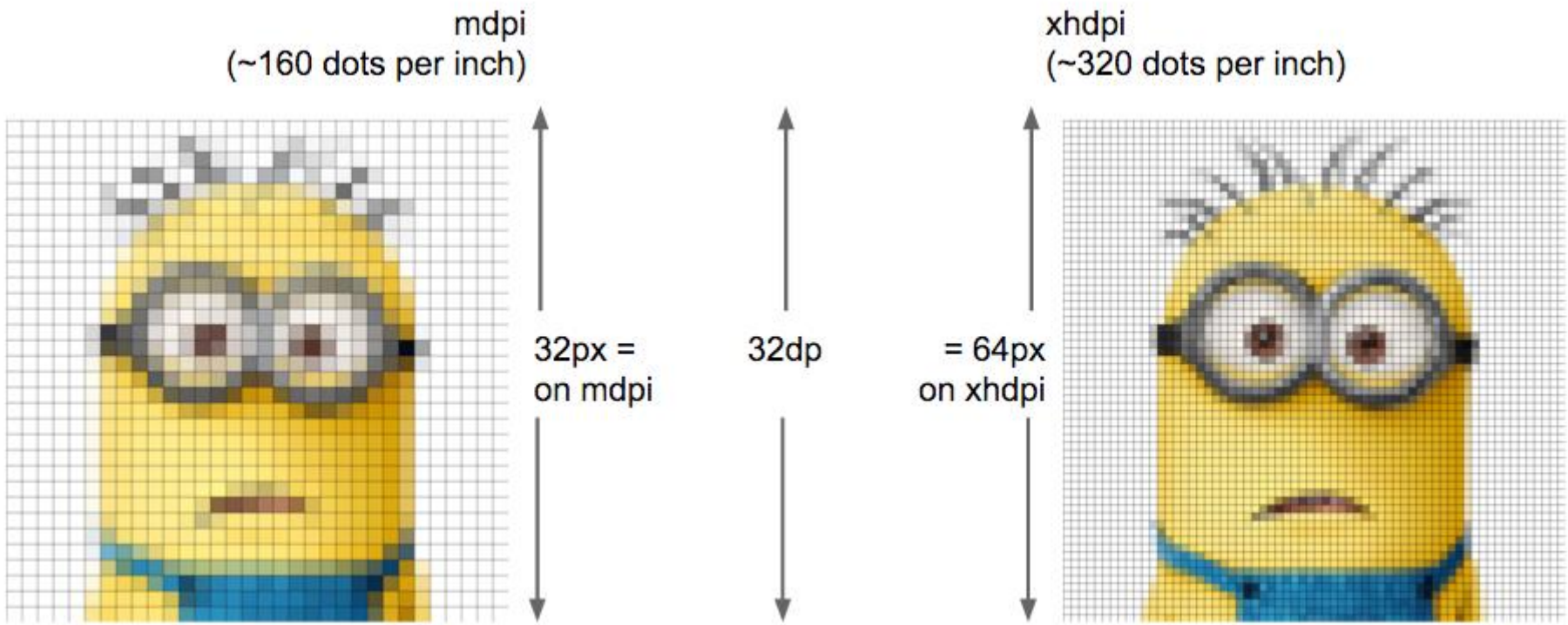


MEASUREMENTS

Dimension	Description	Units / Physical Inch	Density Independent	Same Physical Size on Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density independent pixels	~160	Yes	No
sp	Scale independent pixels	~160	Yes	No

- Any specification of spacing in an Android layout must be specified using one of the following units of measurement:
  - **in** – Inches.
  - **mm** – Millimeters.
  - **pt** – Points (1/72 of an inch).
  - **dp** – Density-independent pixels. An abstract unit of measurement based on the physical density of the device display relative to a 160dpi display baseline.
  - **sp** – Scale-independent pixels. Similar to dp but scaled based on the user's font preference.
  - **px** – Actual screen pixels. Use is not recommended since different displays will have different pixels per inch. Use dp in preference to this unit.

THE VALUE OF EXTRA DPI



MANUALLY CREATING AN XML LAYOUT

- Note that in the following the layout element is also configured with padding on each side of 16dp (density independent pixels).
- **Note**: padding is the space that's inside the element between the element and the border. Padding goes around all four sides of the content and you can target and change the padding for each side.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

MANUALLY CREATING AN XML LAYOUT

- Any children that need to be added to the ConstraintLayout parent must be nested within the opening and closing tags. In the following example a Button widget has been added as a child of the ConstraintLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <Button
        android:text="Click me !!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

MANUALLY CREATING AN XML LAYOUT

- As currently implemented, the button has no constraint connections. At runtime, therefore, the button will appear in the top left-hand corner of the screen (though indented 16dp by the padding assigned to the parent layout). If opposing constraints are added to the sides of the button, however, it will appear centered within the layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <Button
        android:text="Click me !!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

- **Note** that each of the constraints is attached to the element named activity_main which is, in this case, the parent ConstraintLayout instance.

MANUALLY CREATING AN XML LAYOUT

- To add a second widget to the layout, simply embed it within the body of ConstraintLayout element. The following modification, for example, adds a TextView widget to the layout:

```
<Button
    android:text="Click me !!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:text="Hello World from Kotlin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
```

MANUALLY CREATING AN XML LAYOUT

- Once again, the absence of constraints on the newly added TextView will cause it to appear in the top left-hand corner of the layout at runtime. The following modifications add opposing constraints connected to the parent layout to center the widget horizontally, together with a constraint connecting the bottom of the TextView to the top of the button with a margin of 72dp:

```
<TextView
    android:text="Hello World from Kotlin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:layout_marginBottom="72dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/button"/>
```

- Also, note that the Button and TextView views have a number of attributes declared. Both views have been assigned IDs and configured to display text strings represented by string resources named button_string and text_string respectively. Additionally, the wrap_content height and width properties have been declared on both objects so that they are sized to accommodate the content (in this case the text referenced by the string resource value).

MANUALLY CREATING AN XML LAYOUT

- To Create a gap between layout elements (textView and button). We will use the margin configuration.
- **Note**: margins and paddings. Both define free space, but margins work outside an element boundaries and paddings inside an element.

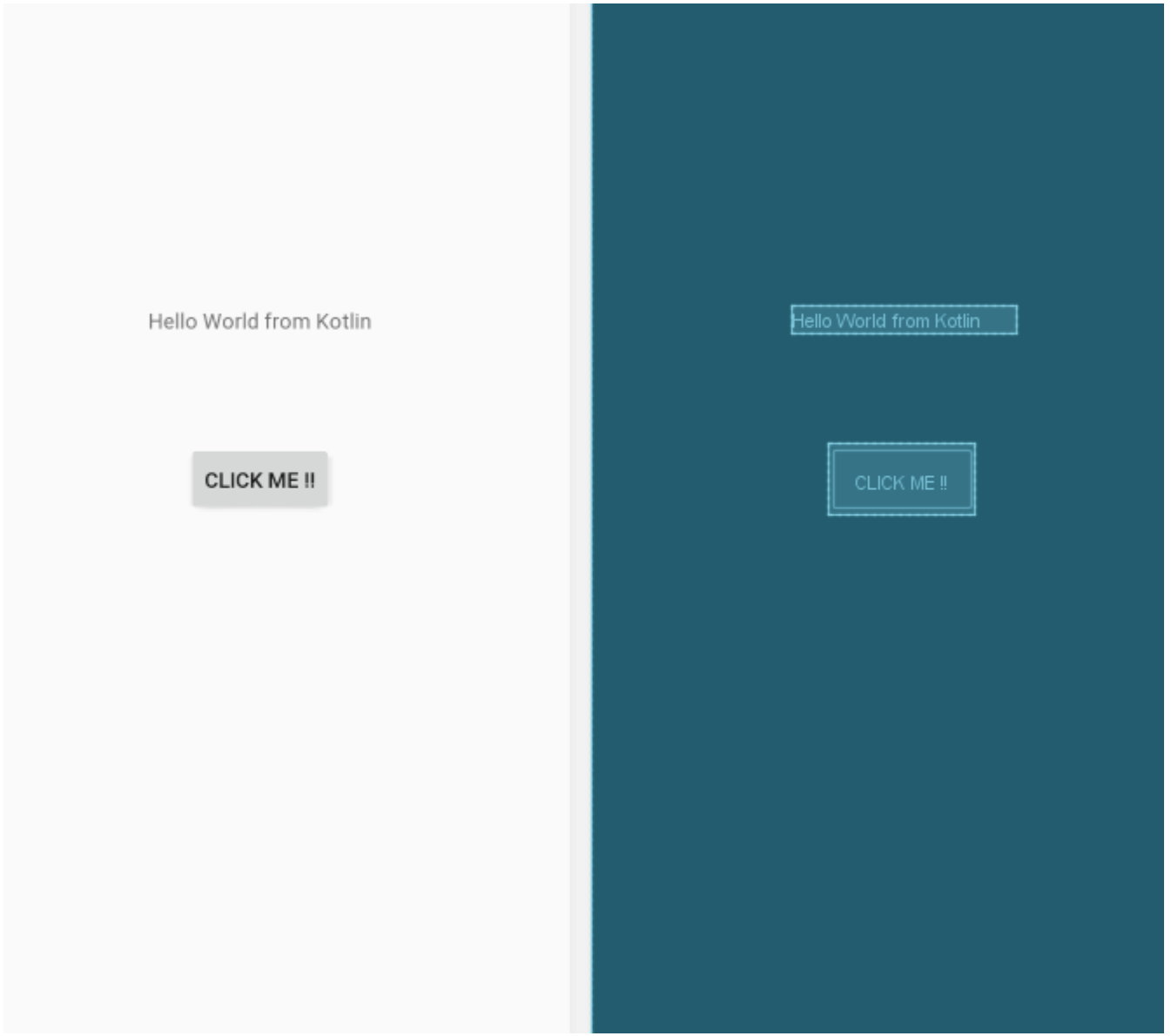
```
<Button
    android:text="Click me !!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:layout_marginBottom="40dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
/>
```

40dp higher than the center

Makes it in the center

MANUALLY CREATING AN XML LAYOUT

- The layout will be rendered as shown in the figure:



MANUALLY CREATING AN XML LAYOUT

To customize individual buttons with a different background, specify the **android:background** attribute with a drawable or color resource.

To customize the text size, we will use **android:textSize** attributes with a specific size.

To change the font type you can use **android:fontFamily** attribute with the font name that you want.

To change the style of your font you can use **android:textStyle** with12 variants are possible: (Regular, Italic, Bold, Bold-italic, Light, Light-italic, Thin, Thin-italic, Condensed regular, Condensed italic, Condensed bold, Condensed bold-italic). **Note**: you can make a combination with this android:textStyle="normal|bold|italic"

MANUALLY CREATING AN XML LAYOUT

- You XML code should be familiar as the following:
- Try to guess android:textColor usage ?!!

```
<Button
    android:text="Click me !!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:background="#EEFF00"
    android:textSize="35dp"
    android:fontFamily="casual"
    android:textStyle="bold"
    android:layout_marginBottom="40dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
/>

<TextView
    android:text="Hello World from Kotlin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:textSize="25dp"
    android:textColor="#9C27B0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/button"
    android:layout_marginBottom="72dp"/>
```

MANUALLY CREATING AN XML LAYOUT

- The layout will be rendered as shown in the figure:

SAMPLE COLOR MAP

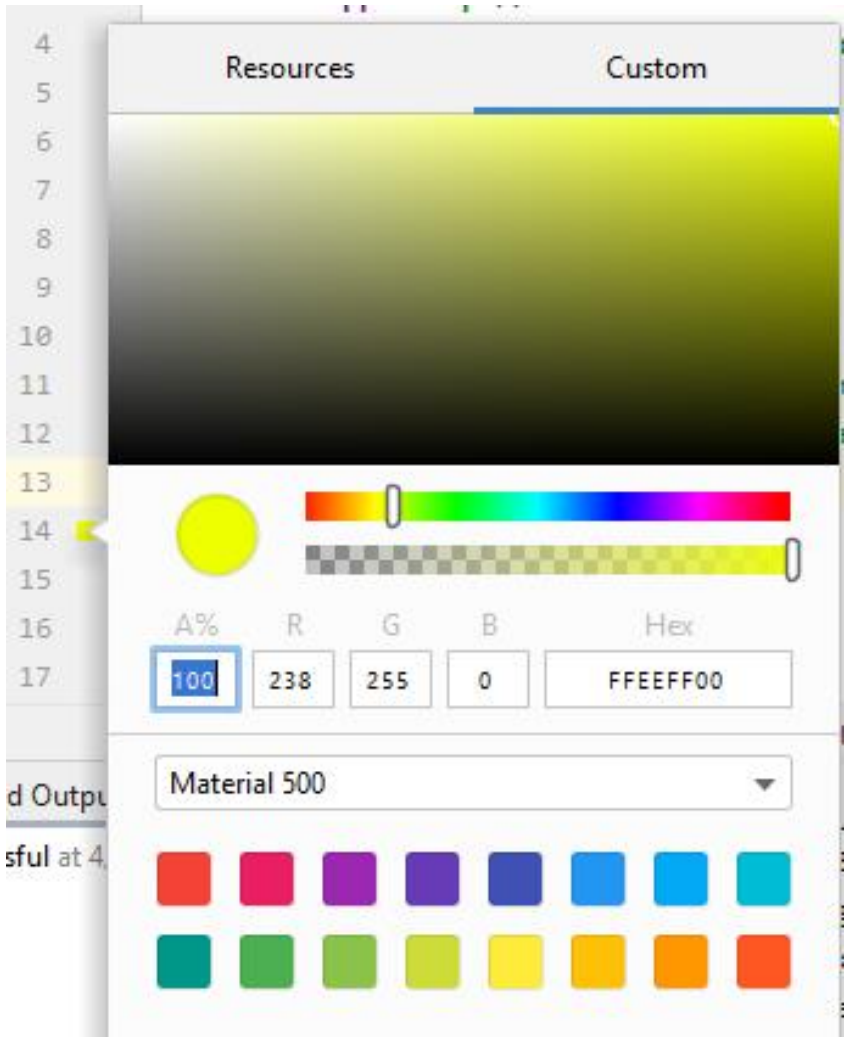
78BFA6	35EC95	0B53A7	F55174	16F52E	A96681	CC6B9F	09B6D5
C264D7	510BFC	61E646	79DA1C	AB7603	5D5C22	384433	194F09
ABFEA4	DA9A48	C5CEDC	09FBF2	B79F81	92C807	CE1685	E4BA38
39DB5E	9964D8	6E0EAD	761313	7FAAF6	A308F5	68FDDC	64B635
86F445	DB9240	362AE7	0FE00E	4D00BE	634588	B53588	5BF76D
363B85	99F2D8	3CE05B	CE18E6	369973	FA9362	9FC929	166F6B
95797B	CD3FC6	B3D317	37BED2	3C0CDB	D61509	0D63C3	9767B7
887C65	168C7E	9DE86B	E6F818	25E6E9	762F39	44FC16	BB4A3A
E9AB52	979CE2	36395C	AB2DE9	A93A4C	D258DA	A72431	677A0D
573574	216D33	EEDCBB	5E461D	4A2A22	57A8D1	DF34D6	7C4590
7A3E89	FFEDF6	0F14CC	C7CA39	E8D8EE	259641	F30E10	51065C
A31C2E	EB1FD4	0164B4	A1310A	C1E49F	532604	4E05DD	AEF03A

FIND HEX OF COLORS TROUGH ANDROID STUDIO

- Write a simple code with a random color, then click on the colored rectangle that appears near the line with the random color. Click on the rectangle and chose your desired color. The Hex code will be added to your XML code directly

```
9   <Button
10      android:text="Click me !! "
11      android:layout_width="wrap_content"
12      android:layout_height="wrap_content"
13      android:id="@+id/button"
14      android:background="#EEFF00"
15      android:textSize="35dp"
16      android:fontFamily="casual"
17      android:textStyle="bold"
```

Click the rectangle



XML OUTSIDE OF LAYOUT FILES

- Sometimes XML will be used to describe types of data **other** than views in your apps; acting as a kind of index that your code can refer to. This is how most apps will define their color palettes for instance, meaning that there's just one file you need to edit if you want to change the look of your entire app.
- You can find this information in the **colors.xml** file, located in **app > resources > values > colors.xml**, which contains tags that assign different names to various color codes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

- You can then refer to this tag subsequently in your Java code **or** your XML code to refer to that particular kind as the following:

```
android:background="@color/colorAccent"
```

XML OUTSIDE OF LAYOUT FILES

- Another alternative use of XML is in the Android Manifest (**AndroidManifest.xml**). This holds a lot of data describing your app, like the label (the app's name), the icon, and instructions about which activities to load first. This helps launchers display the app correctly on the homescreen, and it's also used by app stores.

- Can you do it by your self !

REFERENCES

- https://www.techotopia.com/index.php/Kotlin_-_Manual_XML_Layout_Design_in_Android_Studio
- http://android4beginners.com/2013/07/lesson-2-2-how-to-use-margins-and-paddings-in-android-layout/
- https://www.pluralsight.com/blog/creative-professional/whats-difference-margin-padding
- https://developer.android.com/guide/topics/ui/controls/button