Create a Birthday Card app

About this codelab

1. Introduction

In this codelab, you will build a simple Android app that displays text. You will be able to position the text on the screen by understanding more about User Interface (UI) components in Android.

Prerequisites

- How to create a new app in Android Studio.
- How to run an app in the emulator or on your Android device.

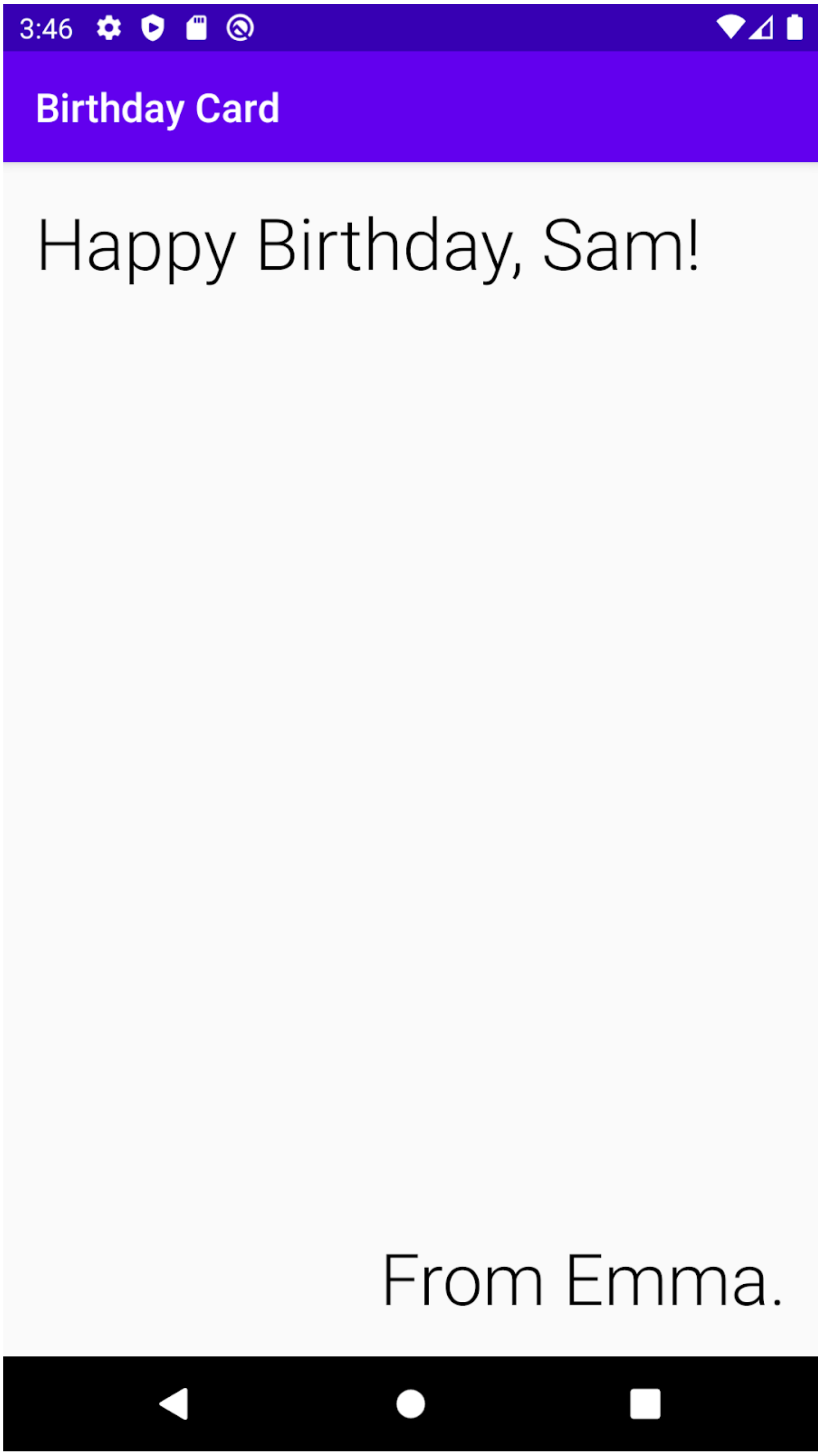
What you will learn

- What are user interface elements, such as `Views` and `ViewGroups`.
- How to display text in a `TextView` in an app.
- How to set attributes, such as text, font, and margin on a `TextView`.

What you will build

- An Android app that displays a birthday greeting in text format.

This is what your app will look like when you're done.



Happy Birthday, Sam!

From Emma.

What you need

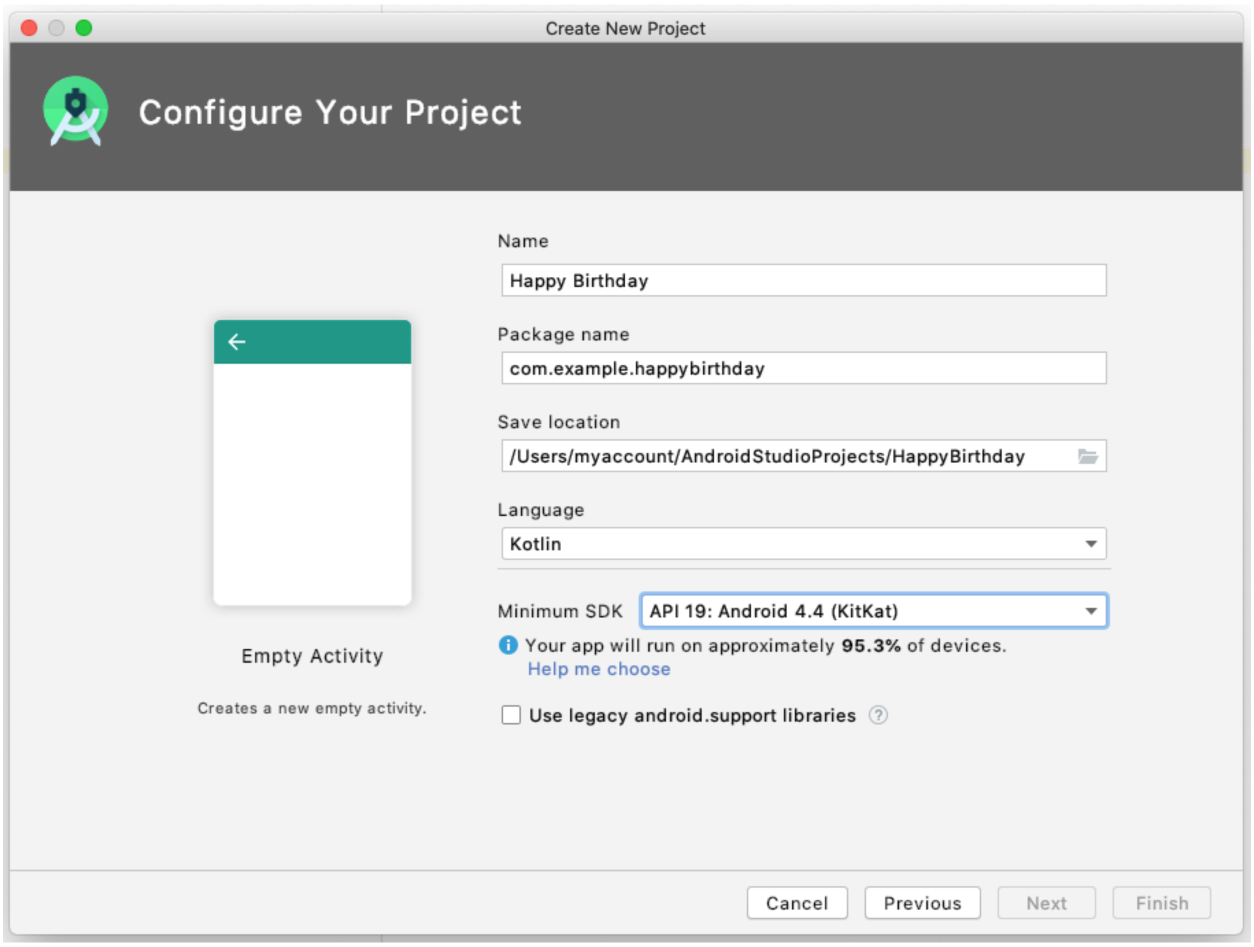
- A computer with Android Studio installed.

2. Set up your Happy Birthday app

Create an Empty Activity project

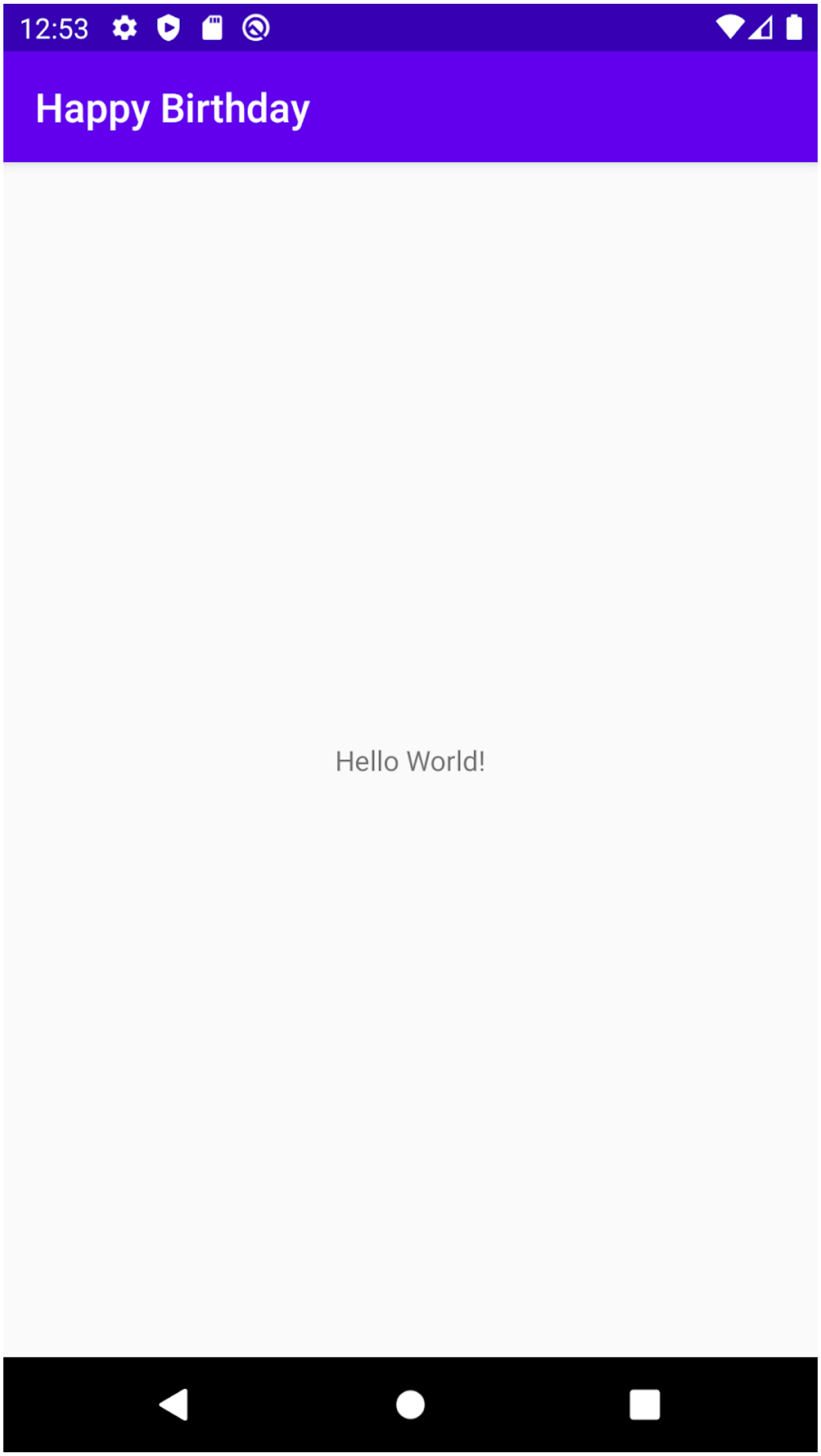
1. To start, create a new Kotlin project in Android Studio using the **Empty Activity** template.
2. Call the app "Happy Birthday", with a minimum API level of 19 (KitKat).

Important: If you're not familiar with creating a new project in Android Studio, see Create and run your first Android app for details.



Name: Happy Birthday
Package name: com.example.happybirthday
Save location: /Users/myaccount/AndroidStudioProjects/HappyBirthday
Language: Kotlin
Minimum SDK: API 19: Android 4.4 (KitKat)
Your app will run on approximately **95.3%** of devices.
Help me choose
☐ Use legacy android.support libraries

3. Run your app. The app should look like the screenshot below.


12:53
Happy Birthday
Hello World!

When you created this Happy Birthday app with the Empty Activity template, Android Studio set up resources for a basic Android application, including a "Hello World!" message in the middle of the screen. In this codelab, you will learn how that message gets there, how to change its text to be more of a birthday greeting, and how to add and format additional messages.

About user interfaces

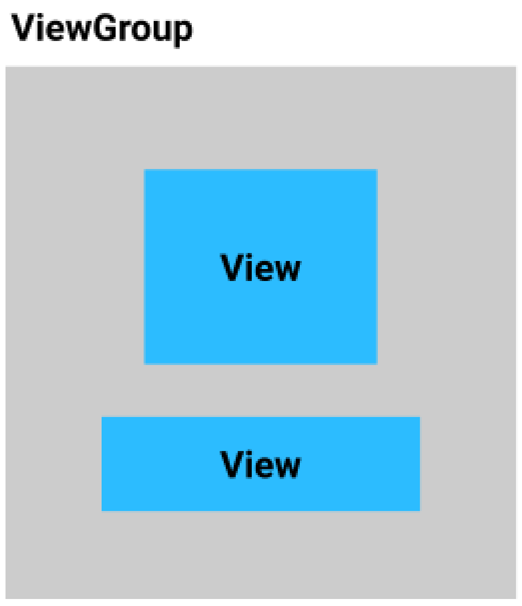
The user interface (UI) of an app is what you see on the screen: text, images, buttons, and many other types of elements. It's how the app shows things to the user, and how the user interacts with the app.

Each of these elements is what's called a `View`. Almost everything you see on the screen of your app is a `View`. `Views` can be interactive, like a clickable button or an editable input field.

In this codelab, you will work with a kind of `View` that is for displaying text and is called a `TextView`.

The `Views` in an Android app don't just float on the screen by themselves. `Views` have relationships to each other. For example, an image may be next to some text, and buttons may form a row. To organize `Views`, you put them in a container. A `ViewGroup` is a container that `View` objects can sit in, and is responsible for arranging the `Views` inside it. The arrangement, or *layout*, can change depending on the size and aspect ratio of the screen of the Android device that the app is running on, and the layout can adapt to whether the device is in portrait or landscape mode.

One kind of `ViewGroup` is a `ConstraintLayout`, which helps you arrange the `Views` inside it in a flexible way.



About the Layout Editor

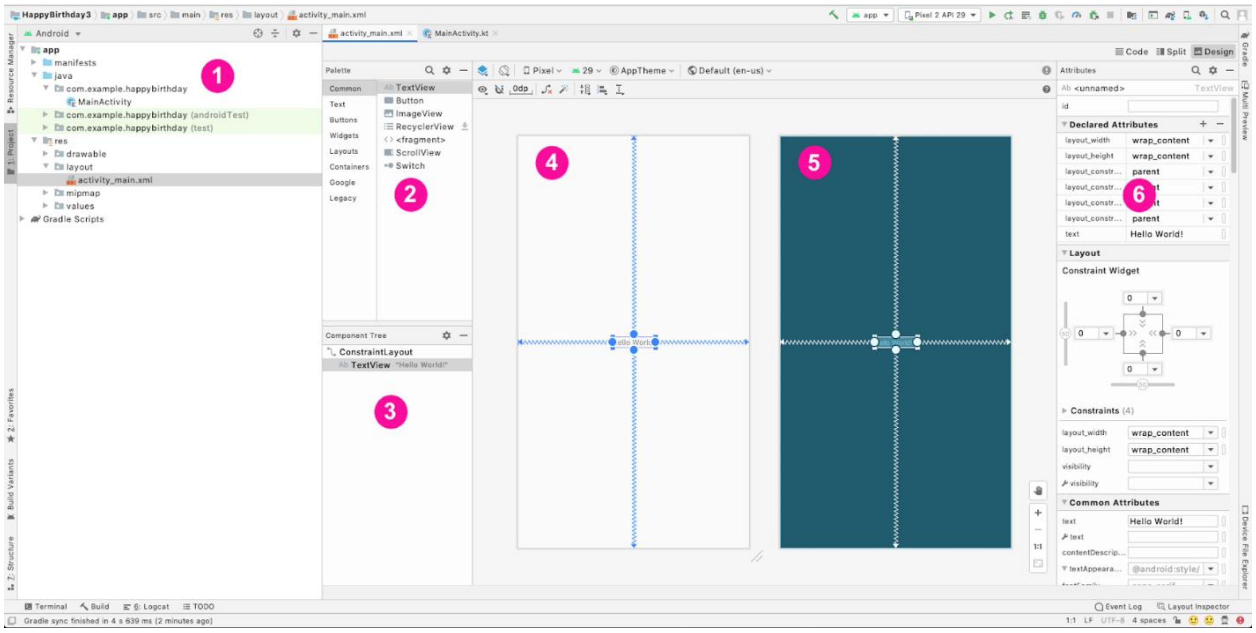
Creating the user interface by arranging `Views` and `ViewGroups` is a big part of creating an Android App. Android Studio provides a tool that helps you do this, called the **Layout Editor**. You'll use the **Layout Editor** to change the "Hello World!" text to "Happy Birthday!", and later, to style the text.

When the **Layout Editor** opens, it has a lot of parts. You will learn to use most of them in this codelab. Use the annotated screenshot below for help recognizing the windows in the **Layout Editor**. You will learn more about each part as you make changes to your app.

- On the left (1) is the **Project** window, which you have seen before. It lists all the files that make up your project.
- At the center you can see two drawings (4) and (5) that represent the screen layout for your app. The representation on the left (4) is a close approximation of what your screen will look like when the app runs. It's called the **Design** view.
- The representation on the right (5) is the **Blueprint** view, which can be useful for specific operations.
- The **Palette** (2) contains lists of different types of `Views` which you can add to your app.
- The **Component Tree** (3) is a different representation of the views of your screen. It lists all the views of your screen.
- On the far right (6) are **Attributes**. It shows you the attributes of a `View` and lets you change them.

Read more about the **Layout Editor** and how to configure it in the developer reference guide.

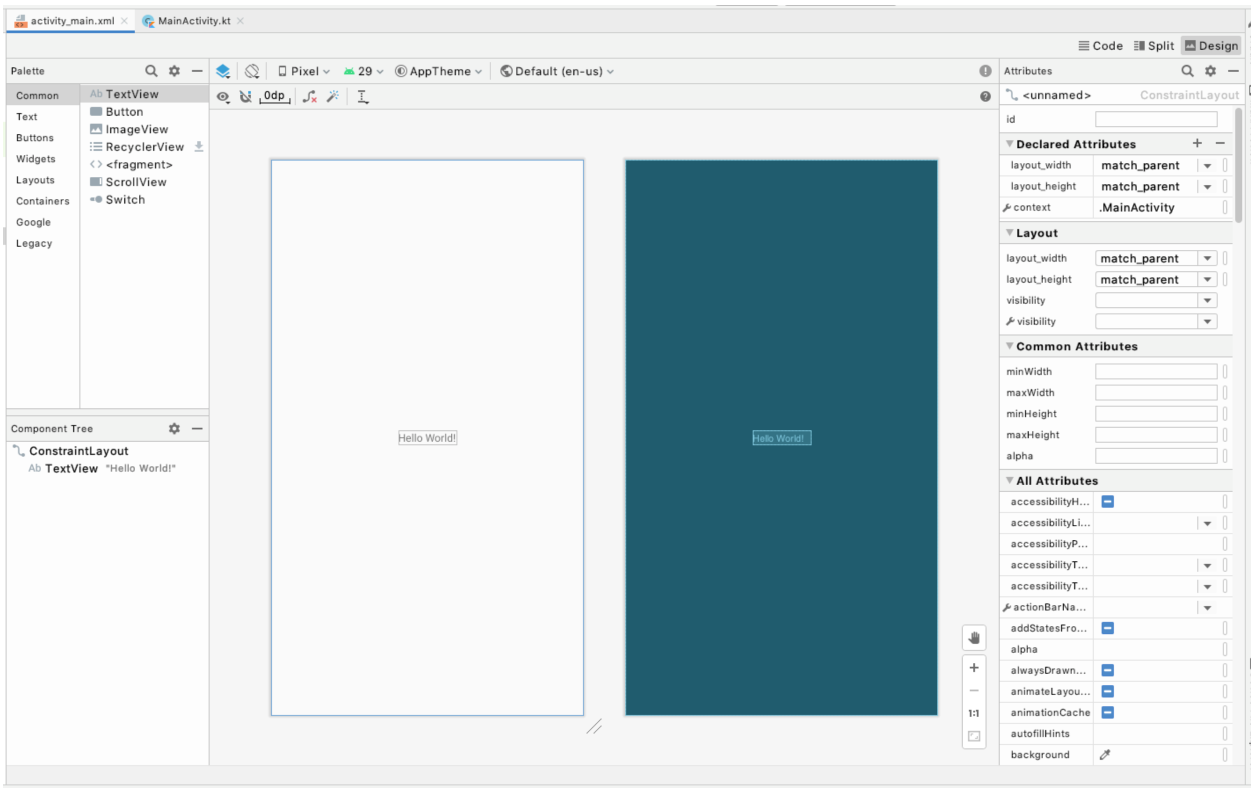
Annotated screenshot of the whole **Layout Editor**:



Let's go make some changes in the **Layout Editor** to make your app more like a birthday card!

Change the Hello World message

1. In Android Studio find the **Project** window on the left.
2. Take note of these folders and files: The **app** folder has most of the files for the app that you will change. The **res** folder is for resources, such as images or screen layouts. The **layout** folder is for screen layouts. The `activity_main.xml` file contains a description of your screen layout.
3. Expand the **app** folder, then the **res** folder, and then the **layout** folder.
4. Double-click on `activity_main.xml`. This opens `activity_main.xml` in the **Layout Editor** and shows the layout it describes in the **Design** view.



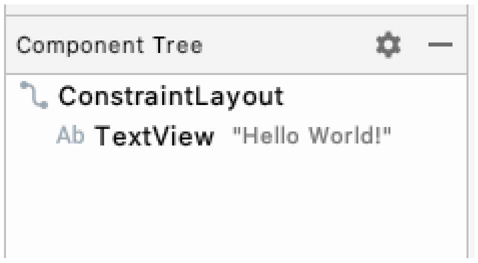
ConstraintLayout, TextView "Hello World!"

<unnamed> ConstraintLayout

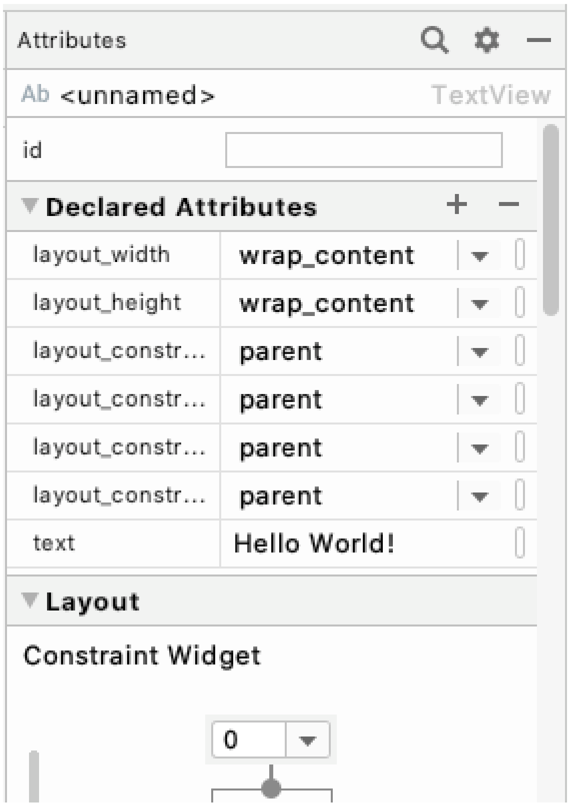
id	
Declared Attributes	
layout_width	match_parent
layout_height	match_parent
context	.MainActivity
Layout	
layout_width	match_parent
layout_height	match_parent
visibility	
visibility	
Common Attributes	
minWidth	
maxWidth	
minHeight	
maxHeight	
alpha	
All Attributes	
accessibilityH...	
accessibilityLi...	
accessibilityP...	
accessibilityT...	
accessibilityT...	
actionBarNa...	
addStatesFro...	
alpha	
alwaysDrawn...	
animateLayou...	
animationCache	
autofillHints	
background	

Note: In these codelabs you will frequently be asked to open a file like in the previous steps. As a shorthand, this may be abbreviated as: Open **activity_main.xml** (**res** > **layout** > **activity_main.xml**) instead of listing each step separately.

5. Look at the list of views in the **Component Tree**. Notice that there is a `ConstraintLayout`, and a `TextView` underneath it. These represent the UI of your app. The `TextView` is indented because it is inside the `ConstraintLayout`. As you add more `Views` to the `ConstraintLayout`, they will be added to this list.
6. Notice that the `TextView` has **"Hello World!"** next to it, which is the text you see when you run the app.



7. In the **Component Tree**, click on the `TextView`.
8. Find **Attributes** on the right.
9. Find the **Declared Attributes** section.
10. Notice that the **text** attribute in the **Declared Attributes** section contains **Hello World!**.

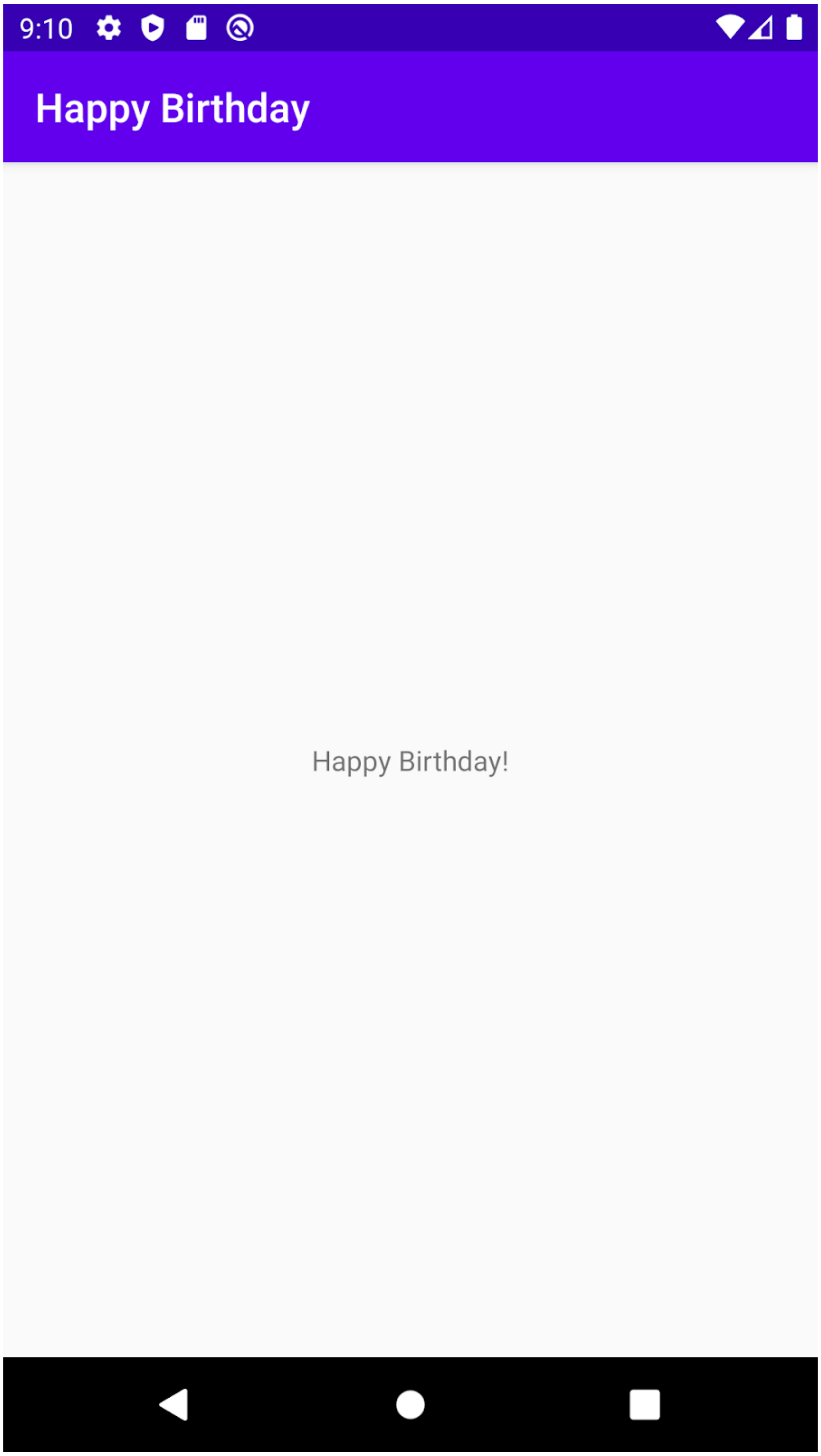


id

layout_width	wrap_content
layout_height	wrap_content
layout_constr...	parent
layout_constr...	parent
layout_constr...	parent
layout_constr...	parent
text	Hello World!

The **text** attribute shows the text that is printed inside a `TextView`.

11. Click on the **text** attribute where the **Hello World!** text is.
12. Change it to say **Happy Birthday!**, then press the **Enter** key. If you see a warning about a hardcoded string, don't worry about it for now. You will learn how to get rid of that warning in the next codelab.
13. Notice that the text has changed in the **Design View**.....(this is cool, you can see your changes right away!)
14. Run your app, and now it says **Happy Birthday!**


9:10
Happy Birthday
Happy Birthday!

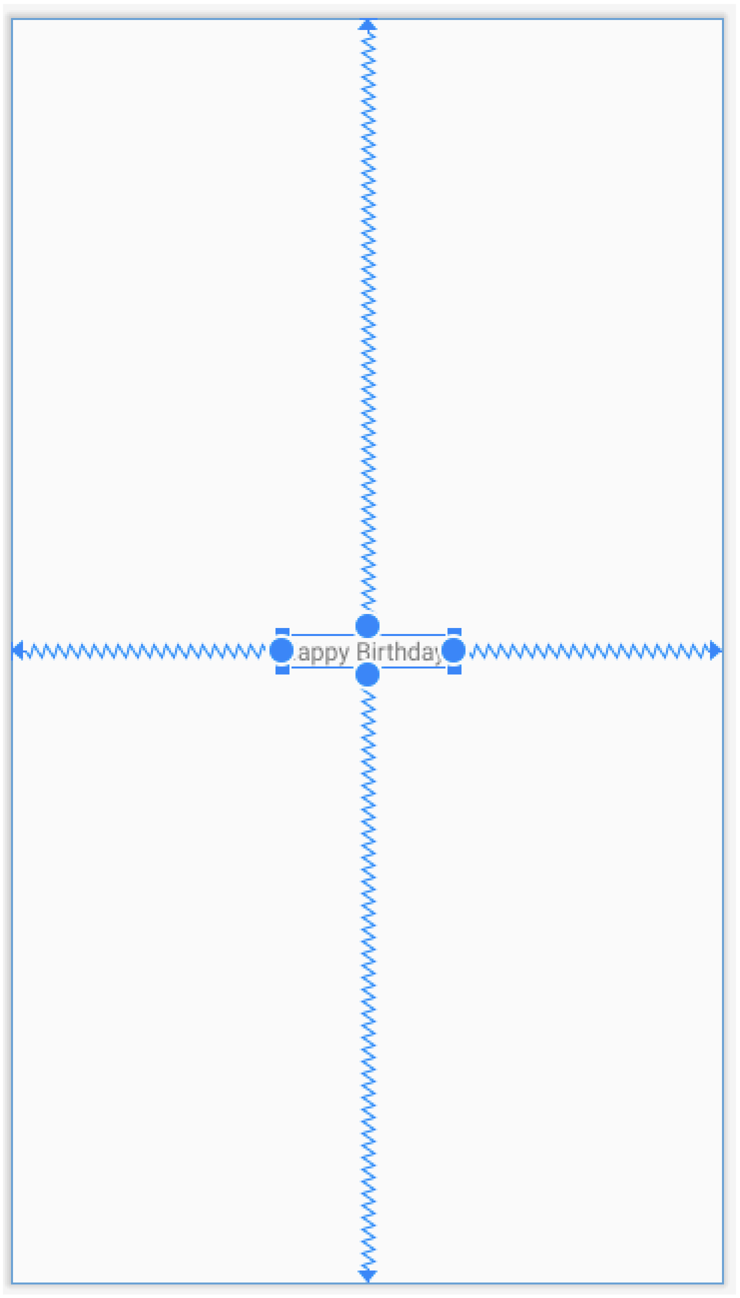
Nice work! You made your first changes to an Android app.

3. Add TextViews to the layout

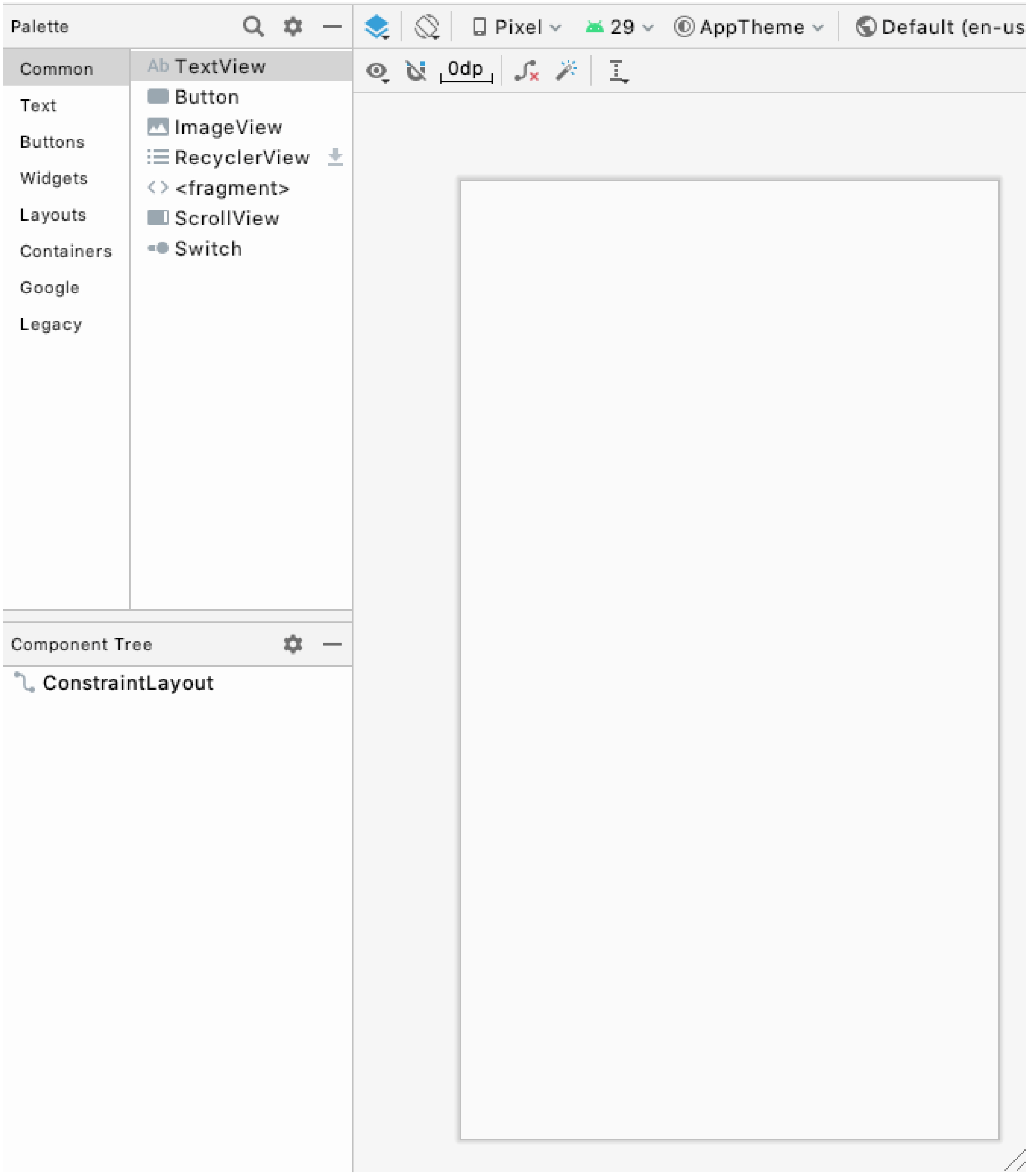
The birthday card you are building looks different than what you have in your app now. Instead of the small text in the center, you need two larger messages, one in the upper left and one in the lower right corner. In this task you'll delete the existing `TextView`, and add two new `TextViews`, and learn how to position them within the `ConstraintLayout`.

Delete the current TextView

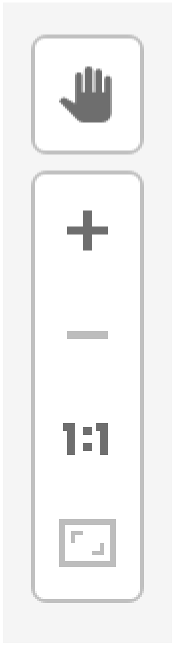
1. In the **Layout Editor**, click to select the `TextView` at the center of the layout.



2. Press the **Delete** key. Android Studio deletes the `TextView`, and your app now shows just a `ConstraintLayout` in the **Layout Editor** and the **Component Tree**.


Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
Pixel
29
AppTheme
Default (en-us
0dp

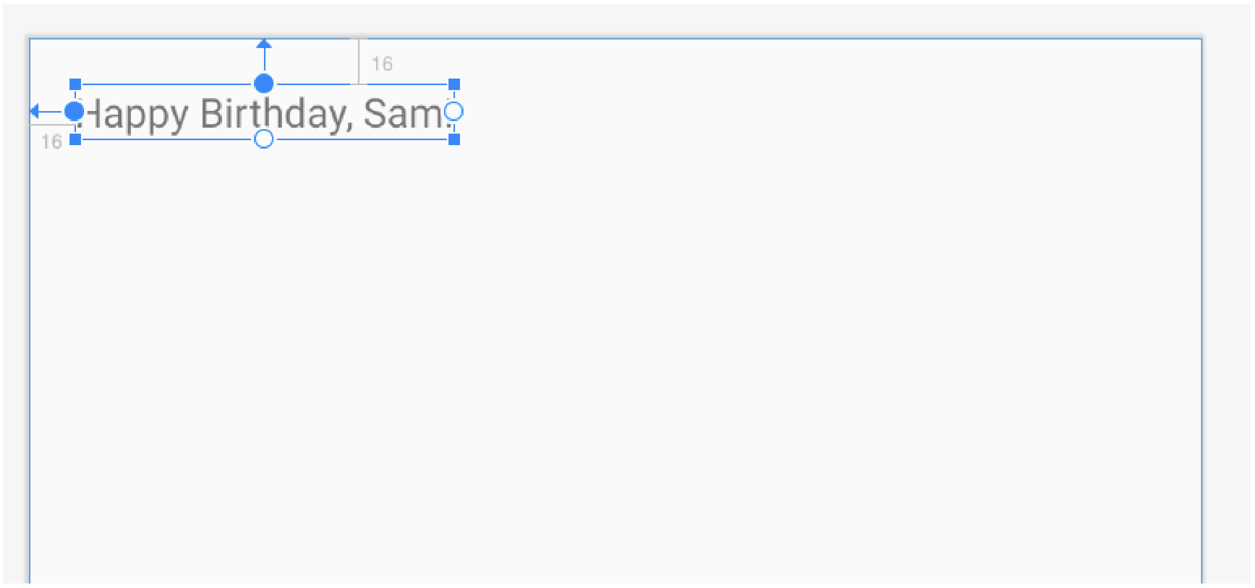
Tip: If you want to zoom in on your layout, you can use the controls at the bottom right of the **Layout Editor** to adjust the size. Click on the bottom-most icon to return to a zoom level where



the whole layout fits on your screen.

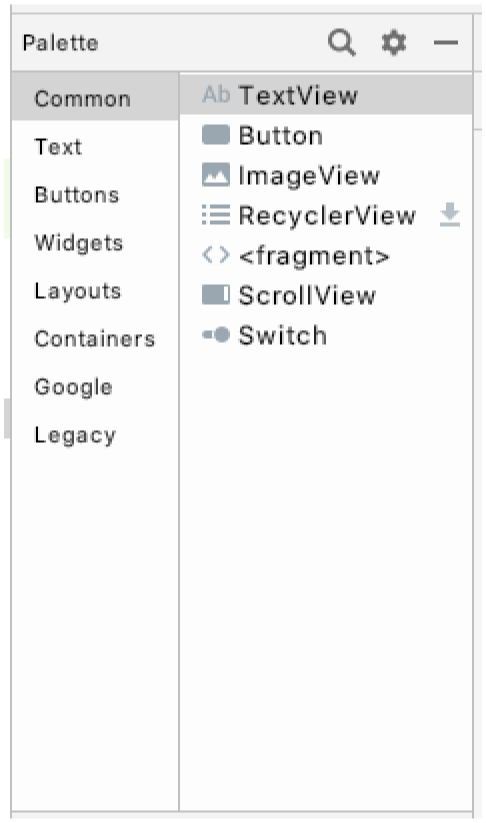
Add a TextView

In this step, you'll add a `TextView` in the upper left of your app to hold the birthday greeting.

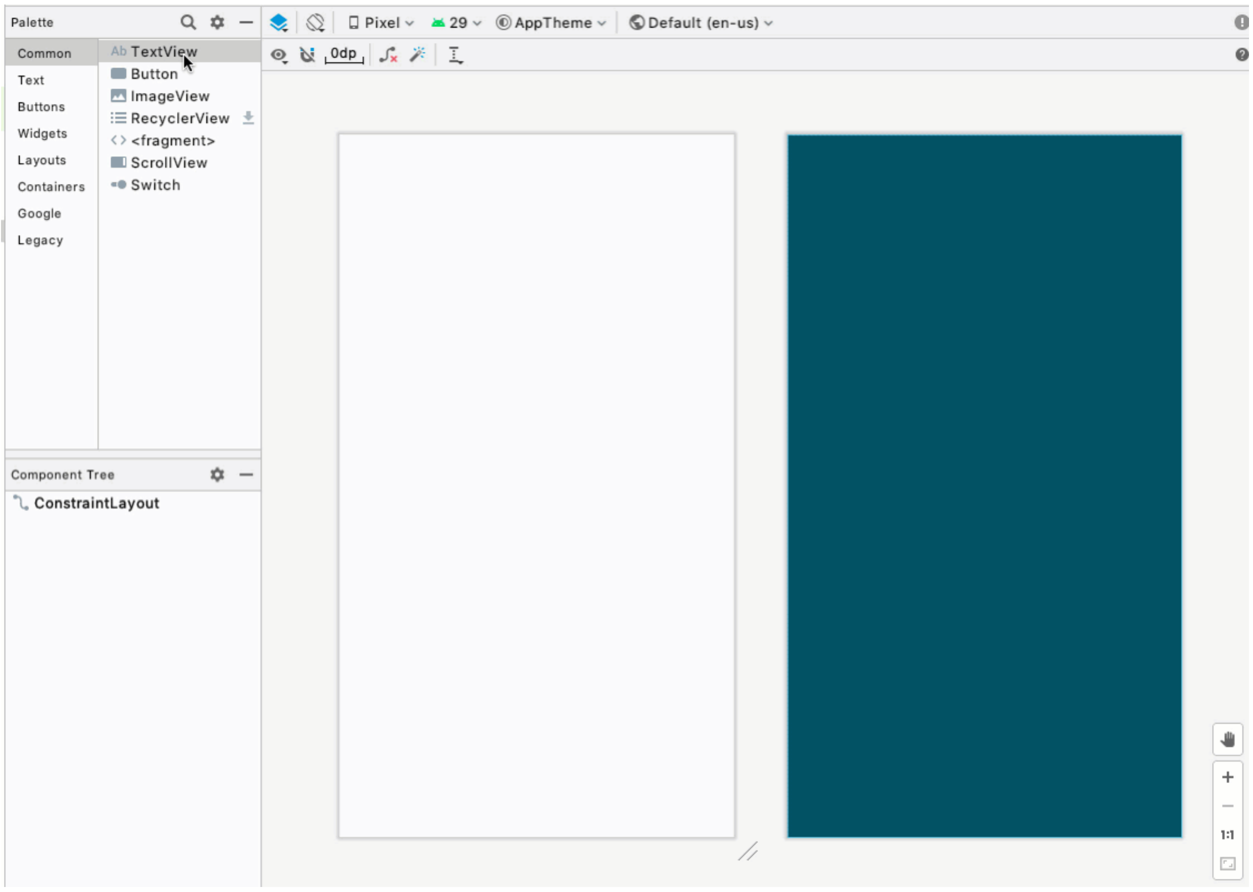


The **Palette** in the upper left of the **Layout Editor** contains lists of different types of `Views`, organized by category, which you can add to your app.

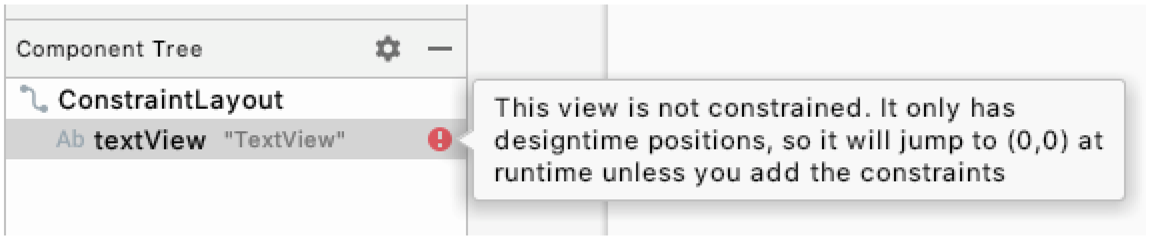
1. Find `TextView`. It appears both in the **Common** category and the **Text** category.



2. Drag a `TextView` from the **Palette** to the upper left of the design surface in the **Layout Editor** and drop it. You don't need to be exact, just drop it near the upper left corner.



3. Notice that there's a `TextView` added, and notice a red exclamation mark in the **Component Tree**.
4. Hover your pointer over the exclamation mark, and you'll see a warning message that the view is not constrained and will jump to a different position when you run the app. You'll fix that in the next step.

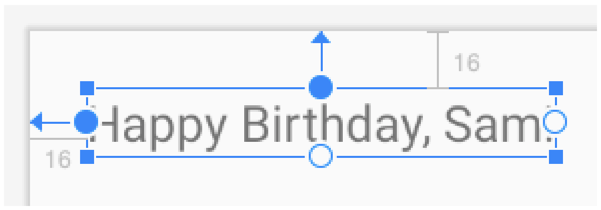


This view is not constrained. It only has designtime positions, so it will jump to (0,0) at runtime unless you add the constraints

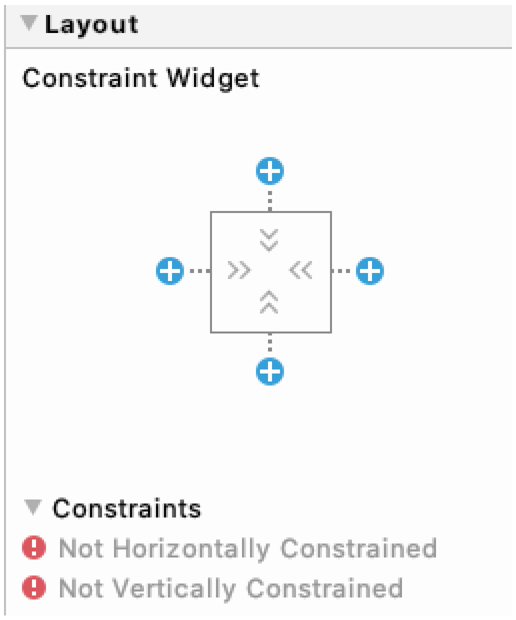
Position the TextView

For the birthday card, the `TextView` needs to be near the upper left corner with some space around it. To fix the warning, you'll add some constraints to the `TextView`, which tell your app how to position it. Constraints are directions and limitations for where a `View` can be in a layout.

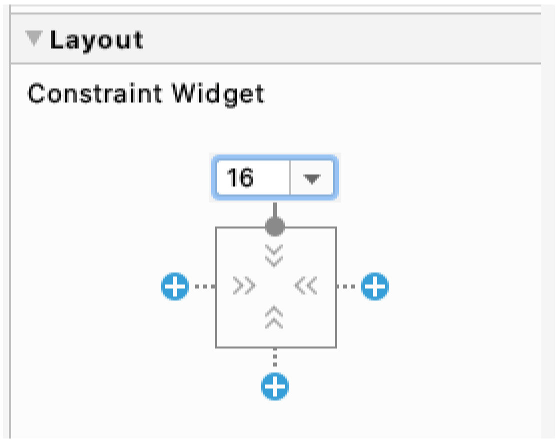
The constraints you add to the top and left will have margins. A margin specifies how far a `View` is from an edge of its container.



1. In **Attributes** on the right, find the **Constraint Widget** in the **Layout** section. The square represents your view.



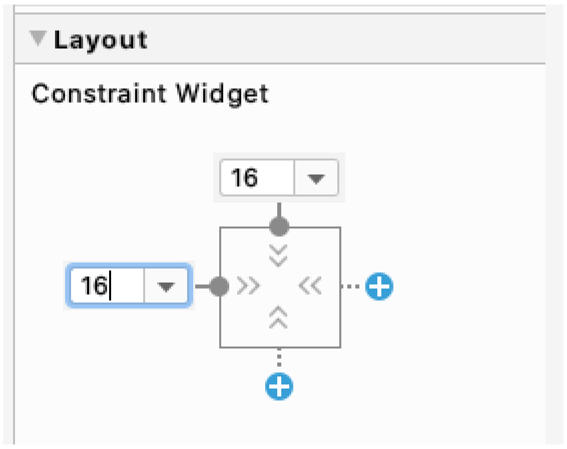
2. Click on the + at the top of the square. This is for the constraint between the top of your text view and the top edge of the constraint layout.
3. A field with a number appears for setting the top margin. The margin is the distance from the `TextView` to the edge of the container, the `ConstraintLayout`. The number you see will be different depending on where you dropped the `TextView`. When you set the top margin, Android Studio automatically also adds a constraint from the top of the text view to the top of the `ConstraintLayout`.



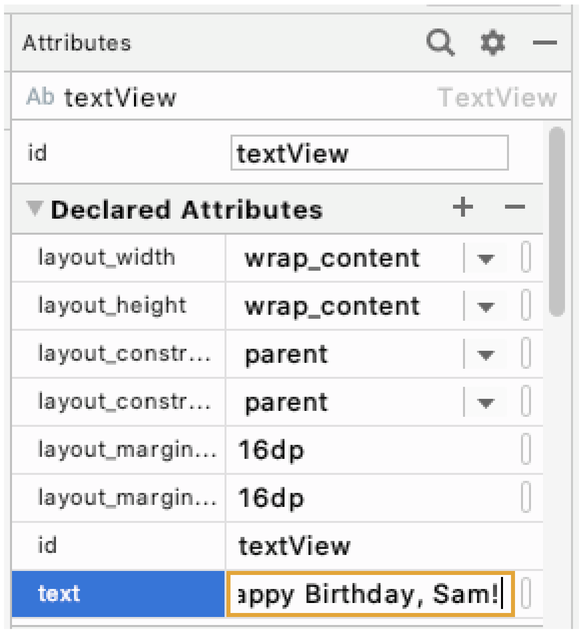
4. Change the top margin to 16.

Note: The unit for margins and other distances in the UI is *density-independent pixels* (dp). It's like centimeters or inches, but for distances on a screen. Android translates this value to the appropriate number of real pixels for each device. As a baseline, 1dp is about 1/160th of an inch, but may be bigger or smaller for some devices.

5. Do the same for the left margin.

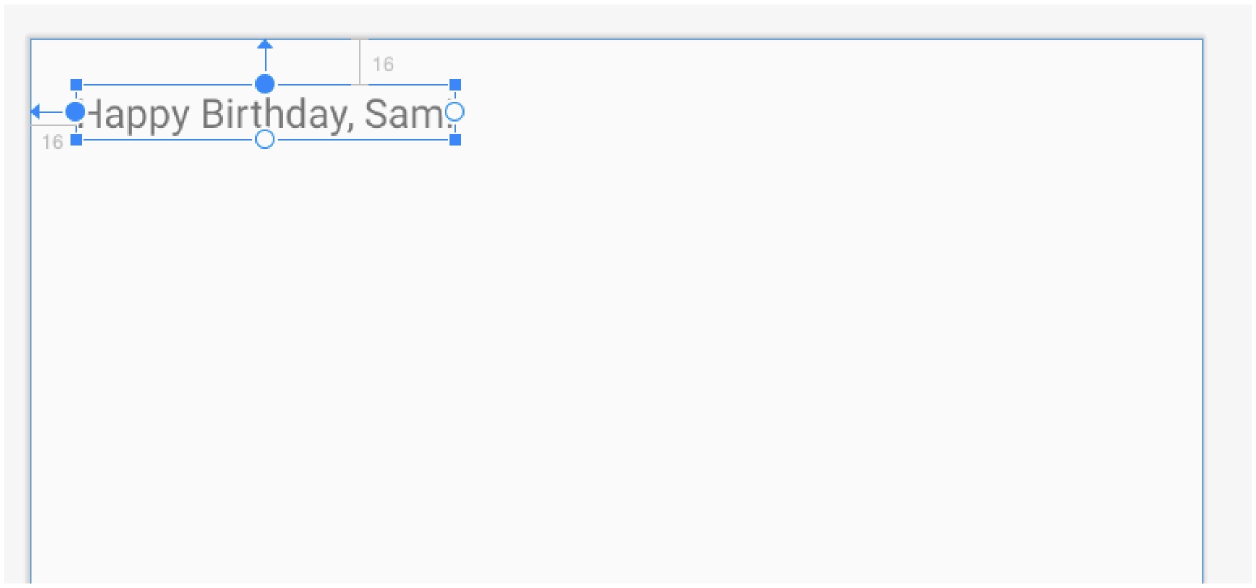


6. Set the **text** to wish your friend a happy birthday, for example "Happy Birthday, Sam!" and press **Enter**.



id	textView
layout_width	wrap_content
layout_height	wrap_content
layout_constr...	parent
layout_constr...	parent
layout_margin...	16dp
layout_margin...	16dp
id	textView
text	appy Birthday, Sam!

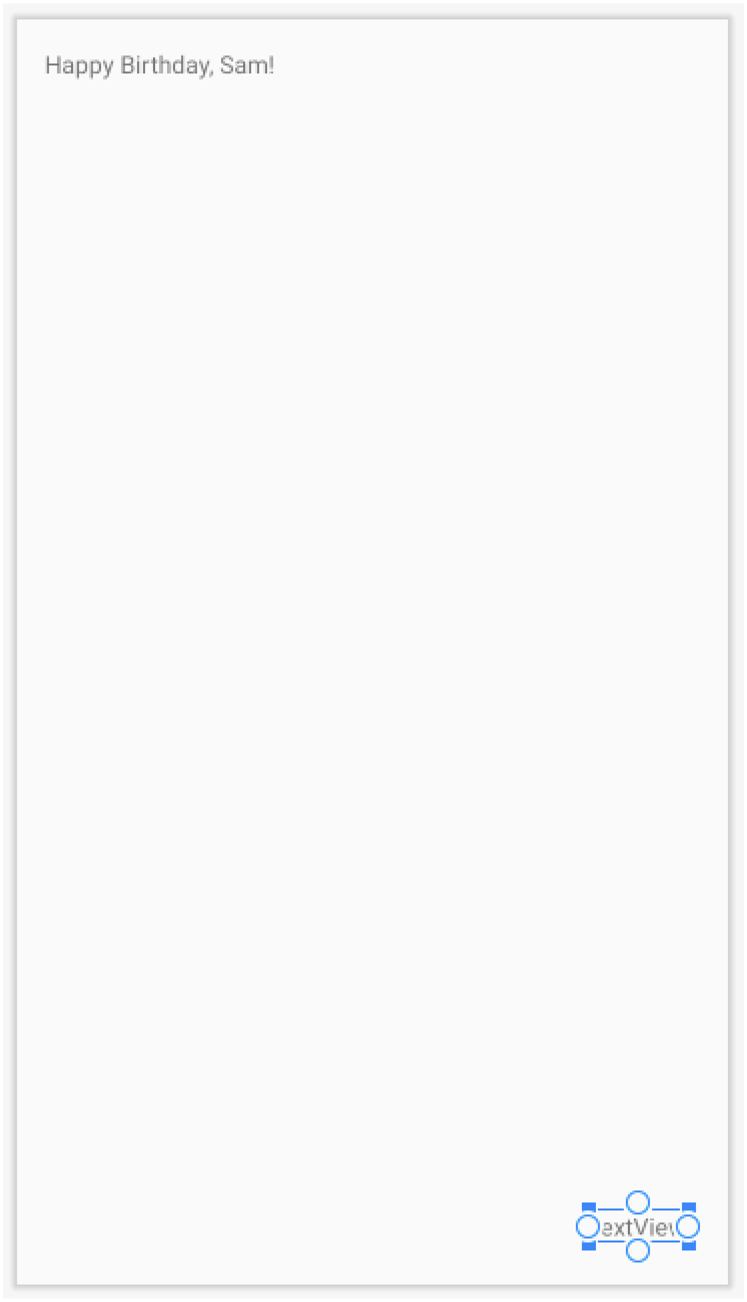
7. Notice that the **Design** view updates to show what your app will look like.



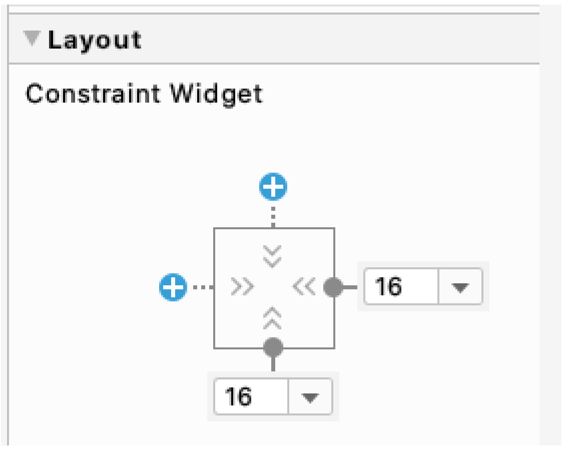
Add and position another TextView

Your birthday card needs a second line of text near the bottom right corner, which you'll add in this step in the same way as in the previous task. What do you think the margins of this `TextView` should be?

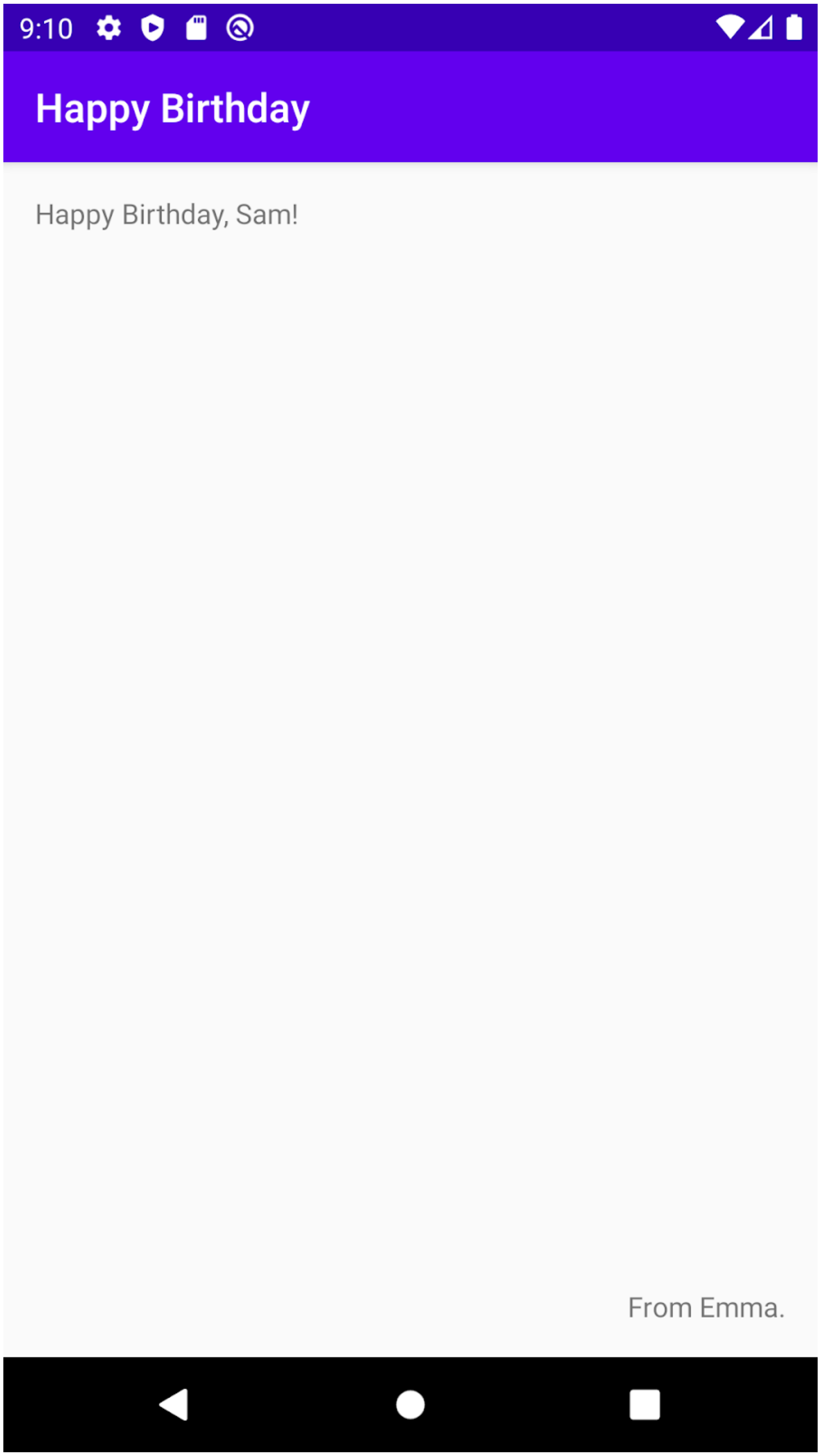
1. Drag a new `TextView` from the **Palette** and drop it near the bottom right of the app view in the Layout Editor.


Happy Birthday, Sam!

2. Set a right margin of 16.
3. Set a bottom margin of 16.



4. In **Attributes**, set the **text** attribute to sign your card, for example "From Emma."
5. Run your app. You should see your birthday wish in the upper left and your signature in the lower right.



Happy Birthday, Sam!

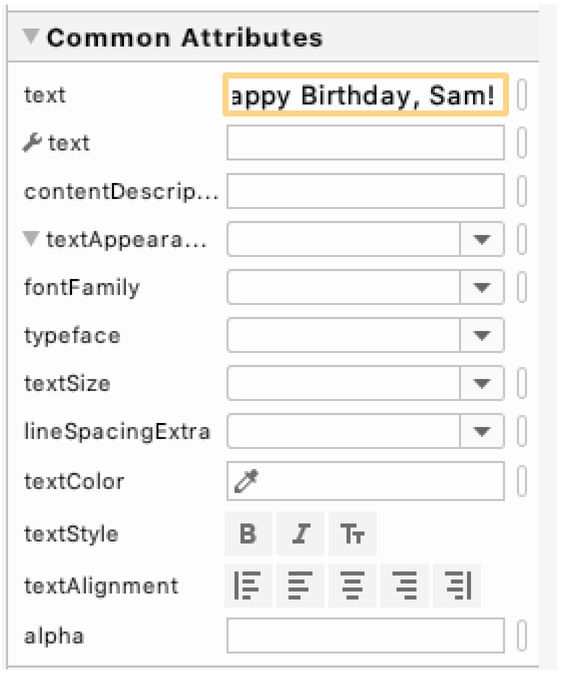
From Emma.

Congratulations! You've added and positioned some UI elements in your app.

4. Add style to the text

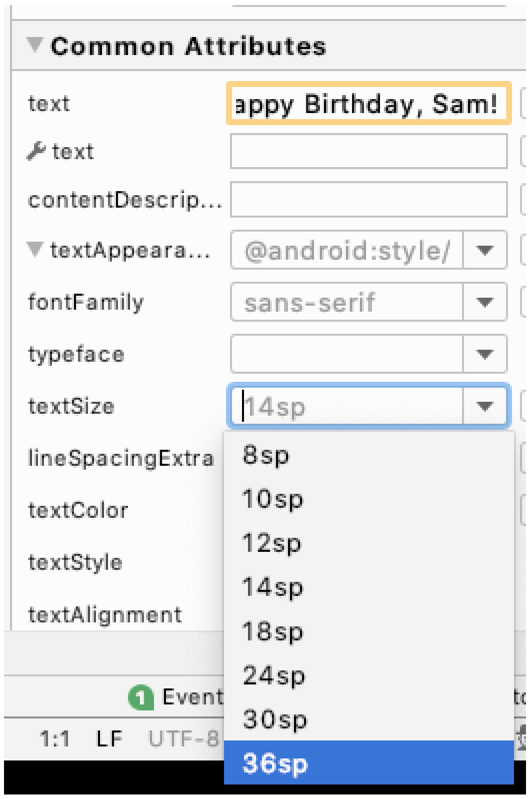
You added text to your user interface, but it doesn't look like the final app yet. In this task, you'll learn how to change the size, text color, and other attributes that affect the appearance of the `TextView`. You can also experiment with different fonts.

1. Click on the first `TextView` in the **Component Tree** and find the **Common Attributes** section of the **Attributes** window. You may need to scroll down to find it.
2. Notice the various attributes including **fontFamily**, **textSize**, and **textColor**.



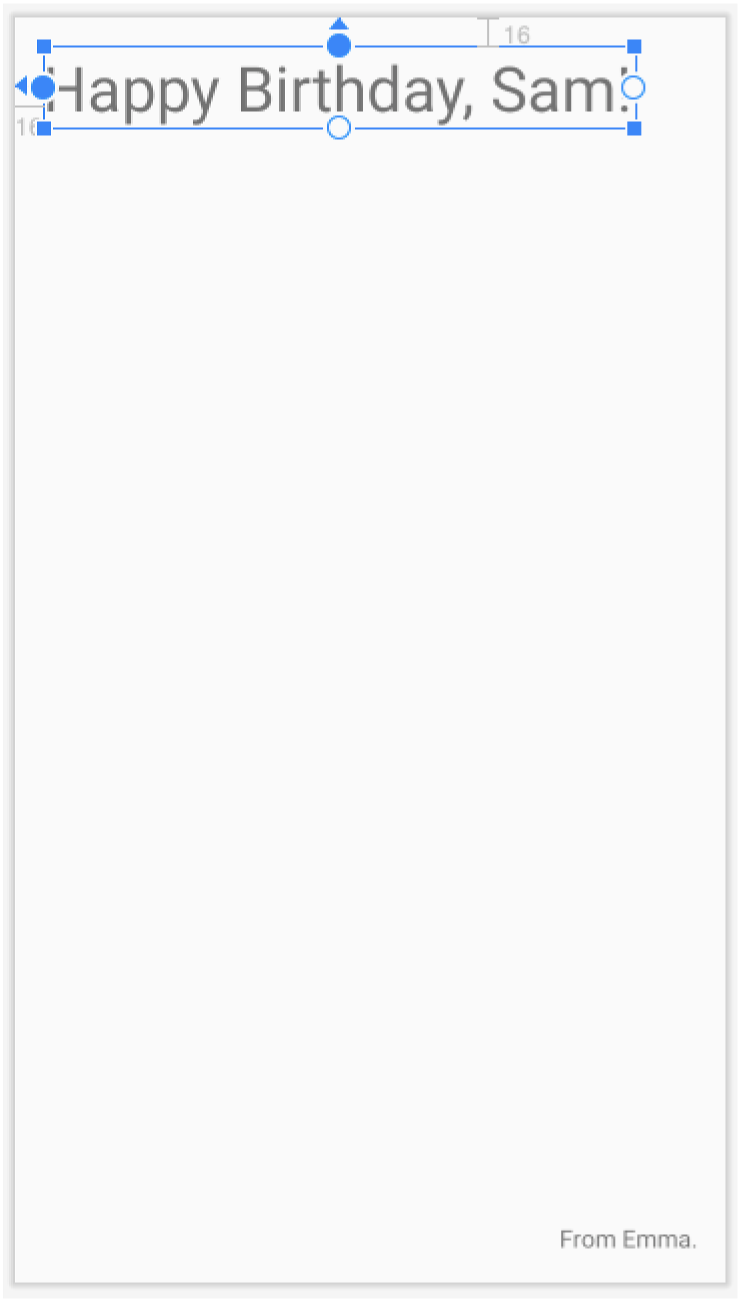
text: appy Birthday, Sam!

3. Look for **textAppearance**.
4. If **textAppearance** is not expanded, click on the down triangle.
5. In the **textSize** set the **textSize** to **36sp**.

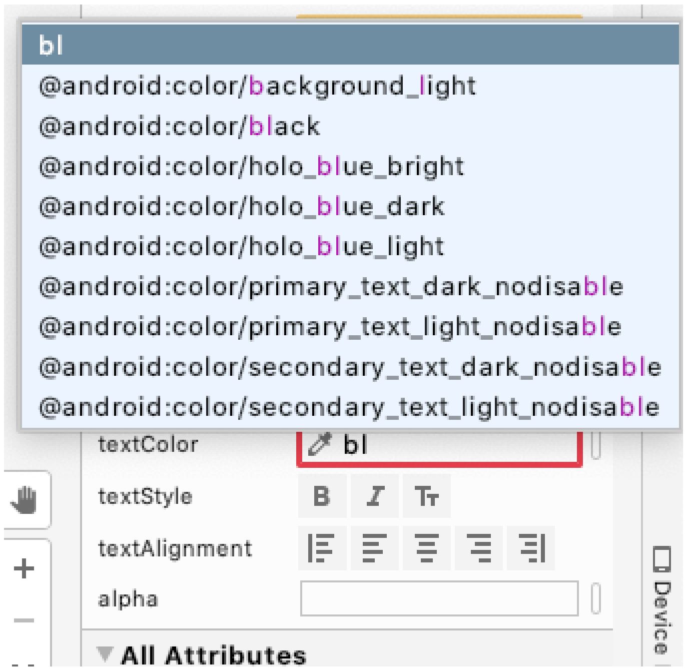


Note: Just like dp is a unit of measure for distances on the screen, **sp** is a unit of measure for the font size. UI elements in Android apps use two different units of measurement, *density-independent pixels* (**dp**) which you used earlier for the layout, and *scalable pixels* (**sp**) which is used when setting the size of text. By default, sp is the same size as dp, but it resizes based on the user's preferred text size.

6. Notice the change in the **Layout Editor**.

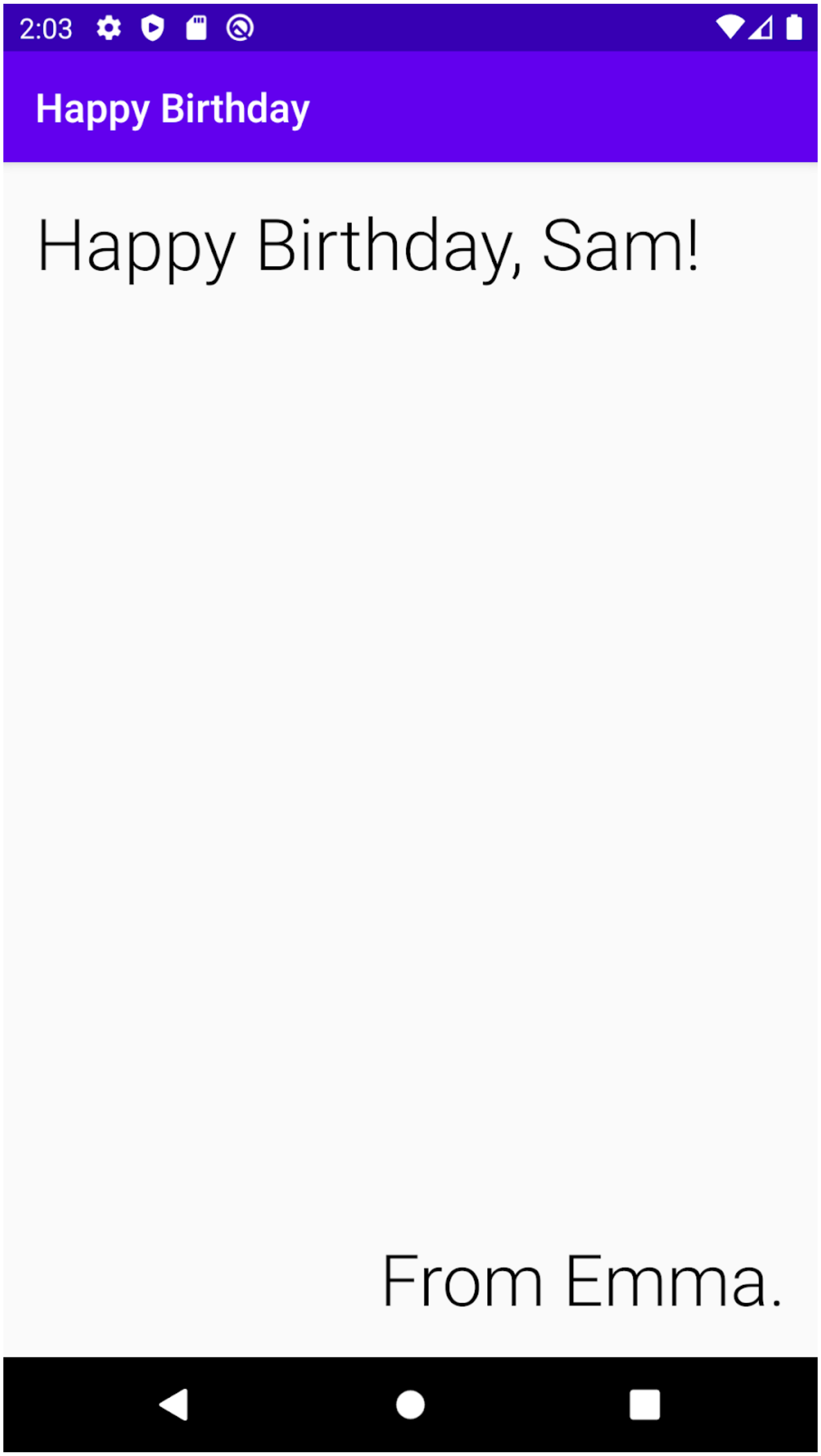


7. Change the **fontFamily** to **casual**.
8. Try some different fonts to see what they look like. There are even more choices for fonts at the bottom of the list, under **More Fonts...**
9. When you're done trying different fonts, set the **fontFamily** to **sans-serif-light**.
10. Click on the **textColor** attribute's edit box and start typing **black**. Notice that as you type, Android Studio shows a list of colors that contain the text you've typed so far.



```
@android:color/background_light
@android:color/black
@android:color/holo_blue_bright
@android:color/holo_blue_dark
@android:color/holo_blue_light
@android:color/primary_text_dark_nodisable
@android:color/primary_text_light_nodisable
@android:color/secondary_text_dark_nodisable
@android:color/secondary_text_light_nodisable
```

11. Select **@android:color/black** from the list of colors and press **Enter**.
12. In the `TextView` with your signature, change the **textSize**, **textColor** and **fontFamily** to match.
13. Run your app and look at the result.



Happy Birthday, Sam!

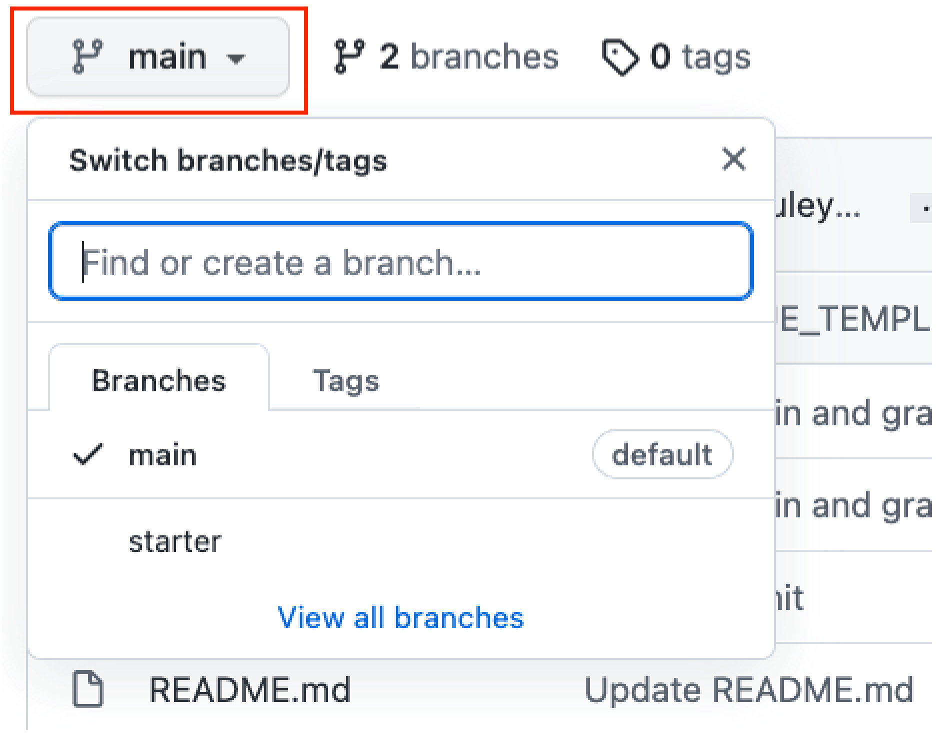
From Emma.

Congratulations, you've taken the first steps to creating a birthday card app!

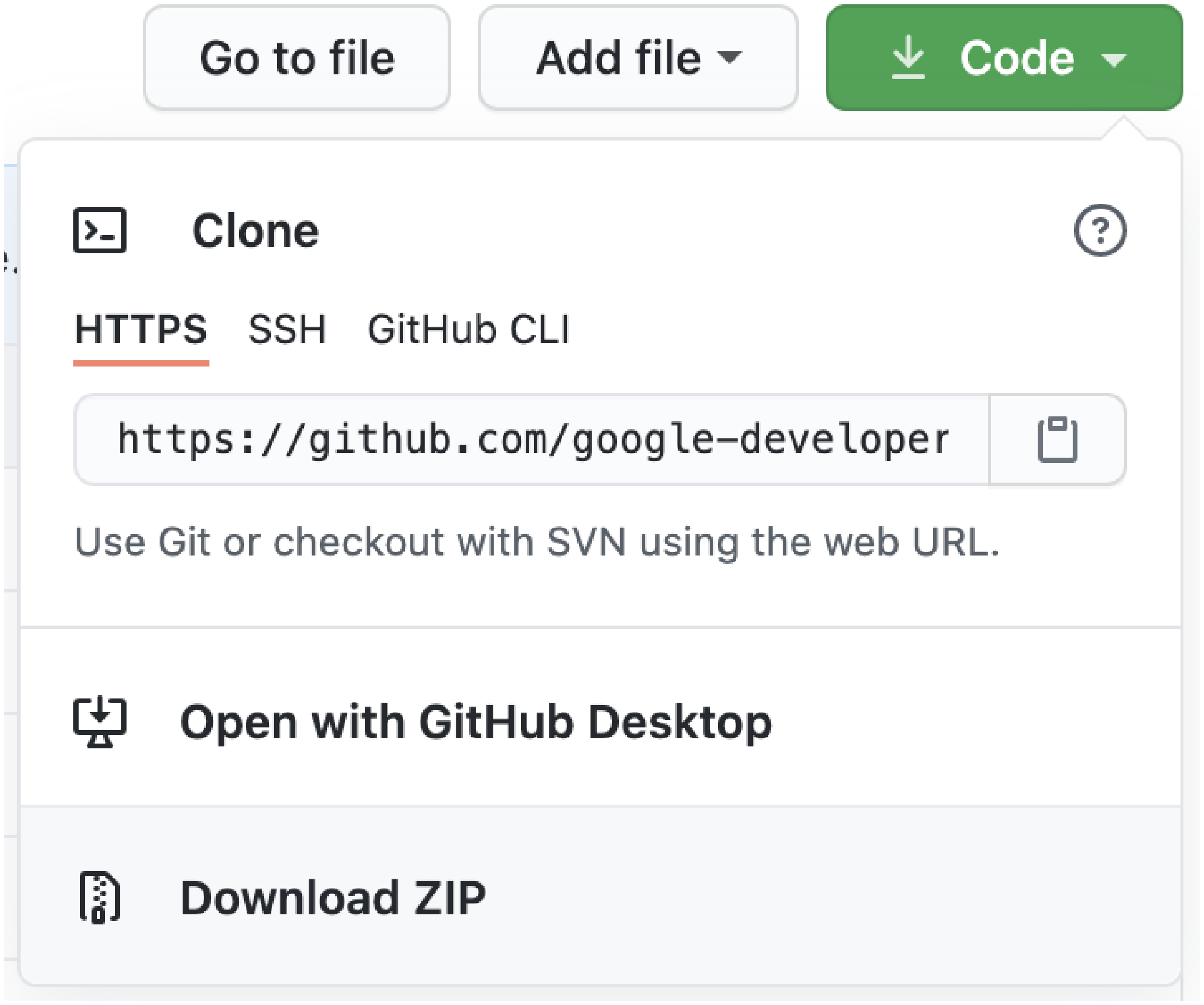
5. Solution

Solution Code URL: https://github.com/google-developer-training/android-basics-kotlin-birthday-card-app-solution

1. Navigate to the provided GitHub repository page for the project.
2. Verify that the branch name matches the branch name specified in the codelab. For example, in the following screenshot the branch name is **main**.



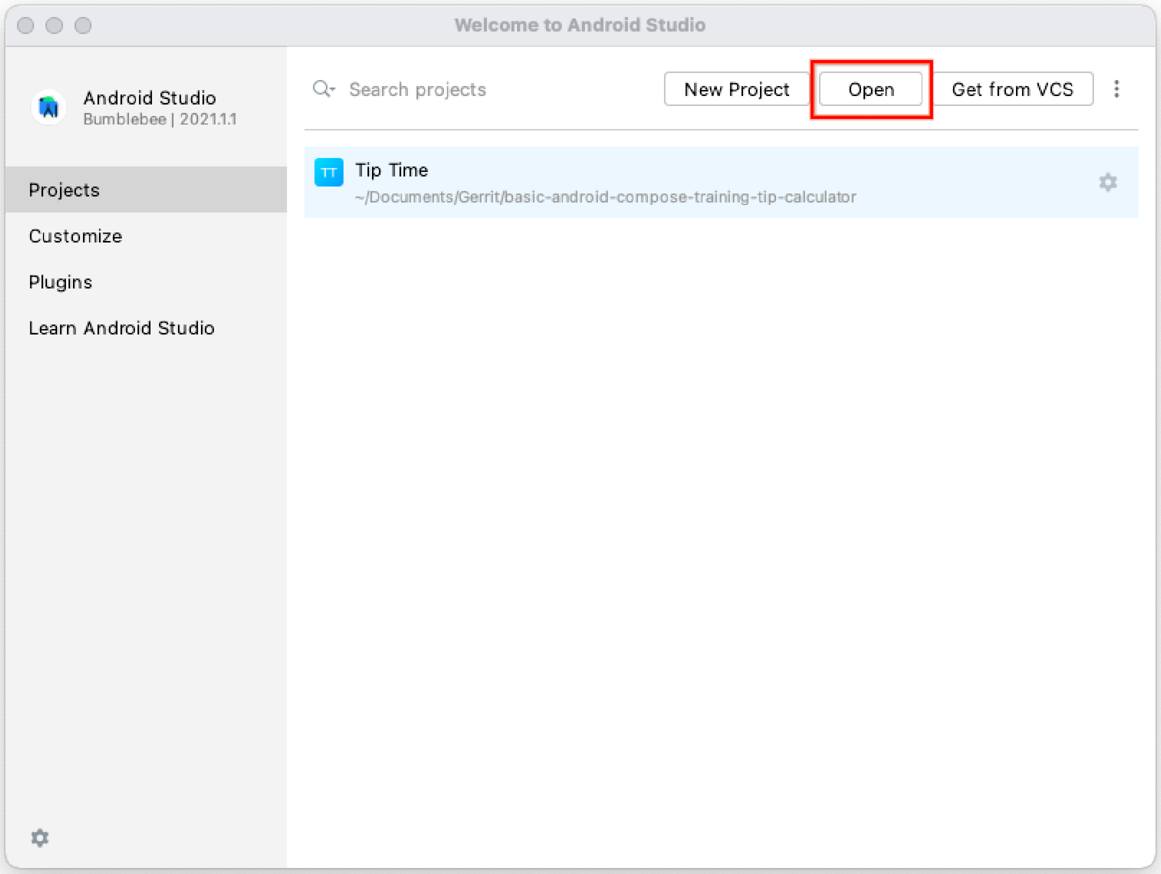
3. On the GitHub page for the project, click the **Code** button, which brings up a popup.


Go to file
Add file
Code
Clone
HTTPS
SSH
GitHub CLI
https://github.com/google-developer
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

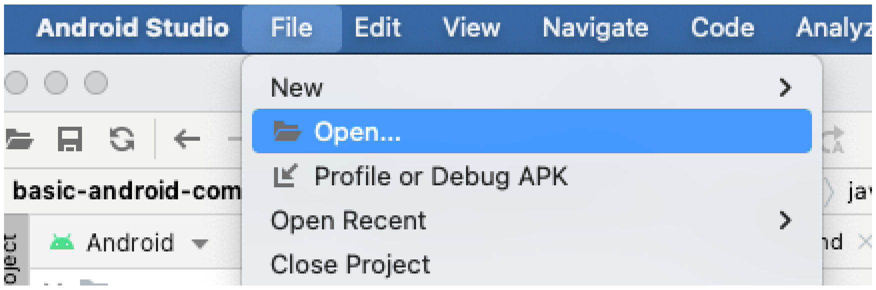
4. In the popup, click the **Download ZIP** button to save the project to your computer. Wait for the download to complete.
5. Locate the file on your computer (likely in the **Downloads** folder).
6. Double-click the ZIP file to unpack it. This creates a new folder that contains the project files.

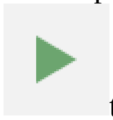
Open the project in Android Studio

1. Start Android Studio.
2. In the **Welcome to Android Studio** window, click **Open**.



Note: If Android Studio is already open, instead, select the **File** > **Open** menu option.



3. In the file browser, navigate to where the unzipped project folder is located (likely in your **Downloads** folder).
4. Double-click on that project folder.
5. Wait for Android Studio to open the project.
6. Click the **Run** button  to build and run the app. Make sure it builds as expected.

Alert: In Android Studio if you see an error, "**Android framework is detected. Click to configure**", the project isn't being detected, or the run button is disabled, ensure that you're opening the sub-directory, **HappyBirthday** in Android Studio and not the parent directory.

6. Summary

- The **Layout Editor** helps you create the UI for your Android app.
- Almost everything you see on the screen of your app is a `View`.
- A `TextView` is a UI element for displaying text in your app.
- A `ConstraintLayout` is a container for other UI elements.
- `Views` need to be constrained horizontally and vertically within a `ConstraintLayout`.
- One way to position a `View` is with a margin.
- A margin says how far a `View` is from an edge of the container it's in.
- You can set attributes on a `TextView` like the font, text size, and color.

7. Learn more

- Vocabulary for Android Basics in Kotlin
- `View`
- `TextView`
- `ConstraintLayout`
- dp vs. sp
- Layout Editor in Android Studio