TRY ANDROID

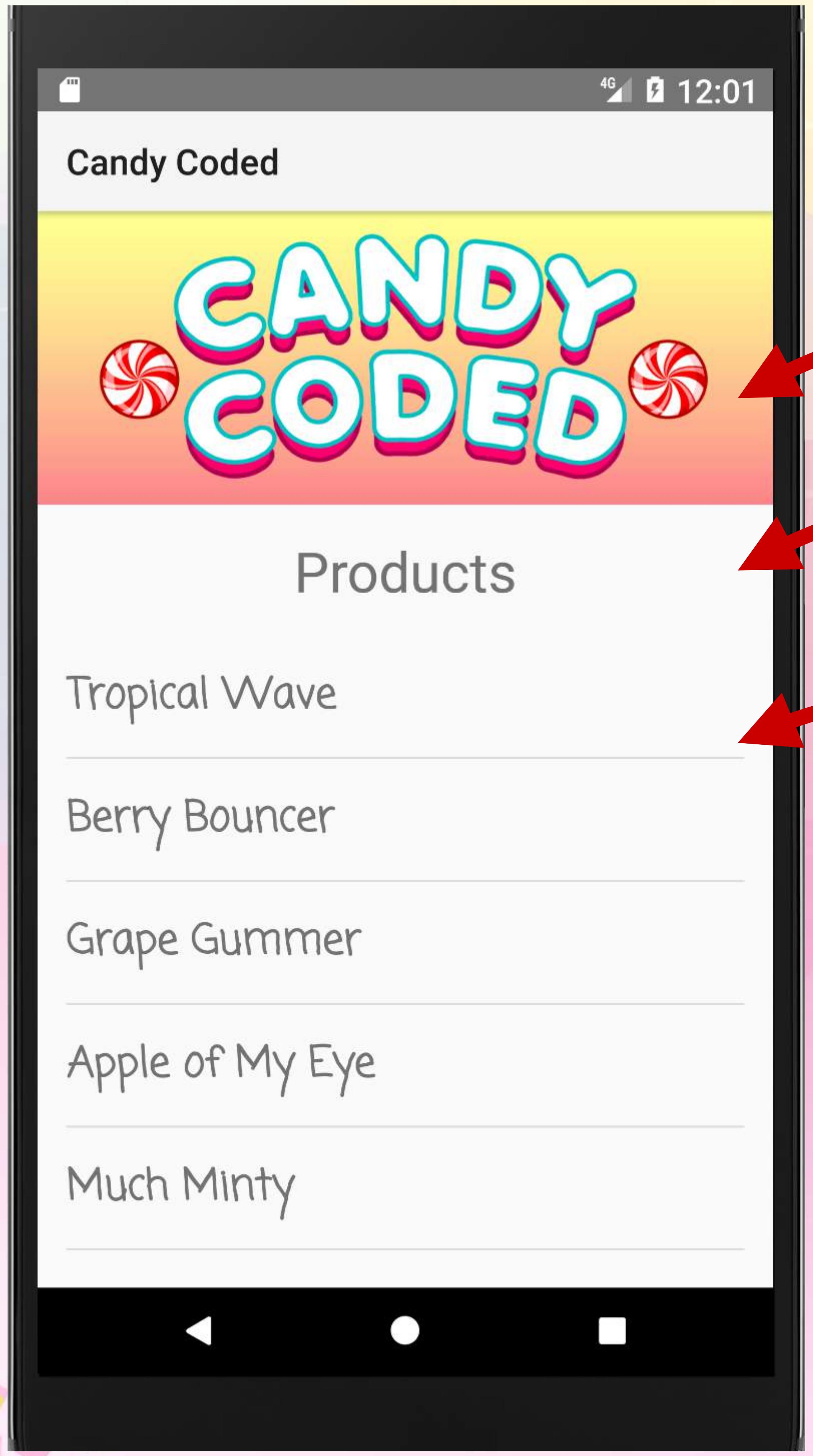
Level 2

Layouts, Views & Images

Creating a LinearLayout & ImageView

Adding an ImageView

The next step in completing our app is adding an ImageView **for our Candy Coded logo.**



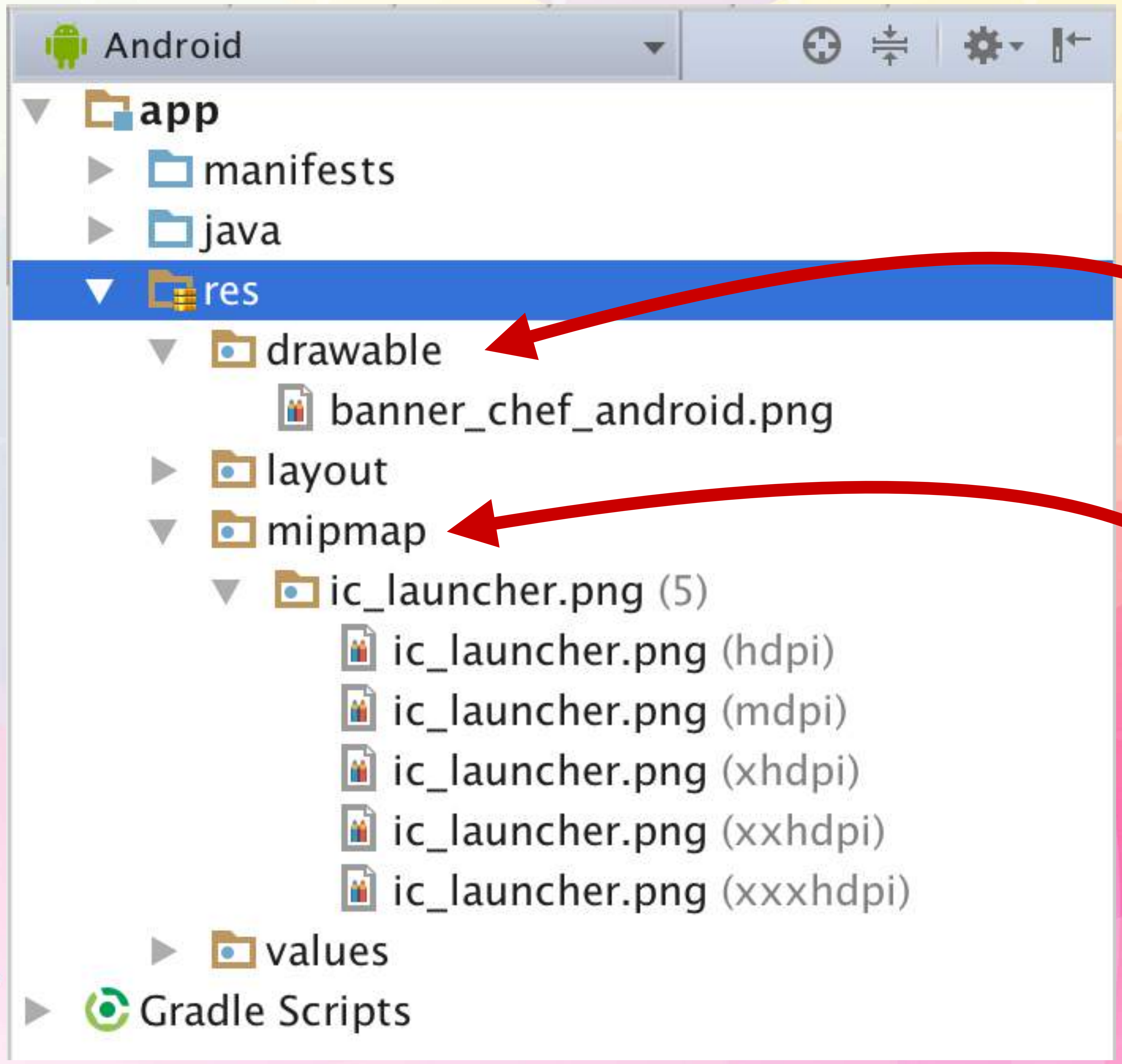
An `ImageView` with our Candy Coded logo

A `TextView` with text "Products" ✓

A `ListView` listing all of our candy products

Android Image Resources

Image resources will either live in the drawable or mipmap folders



```
Android
▼ app
  ▶ manifests
  ▶ java
  ▼ res
    ▼ drawable
        banner_chef_android.png
    ▶ layout
    ▼ mipmap
      ▼ ic_launcher.png (5)
          ic_launcher.png (hdpi)
          ic_launcher.png (mdpi)
          ic_launcher.png (xhdpi)
          ic_launcher.png (xxhdpi)
          ic_launcher.png (xxxhdpi)
    ▶ values
▶ Gradle Scripts
```

Images that are not affected by changes in scale can be put in the `drawable` *folder under* `res`

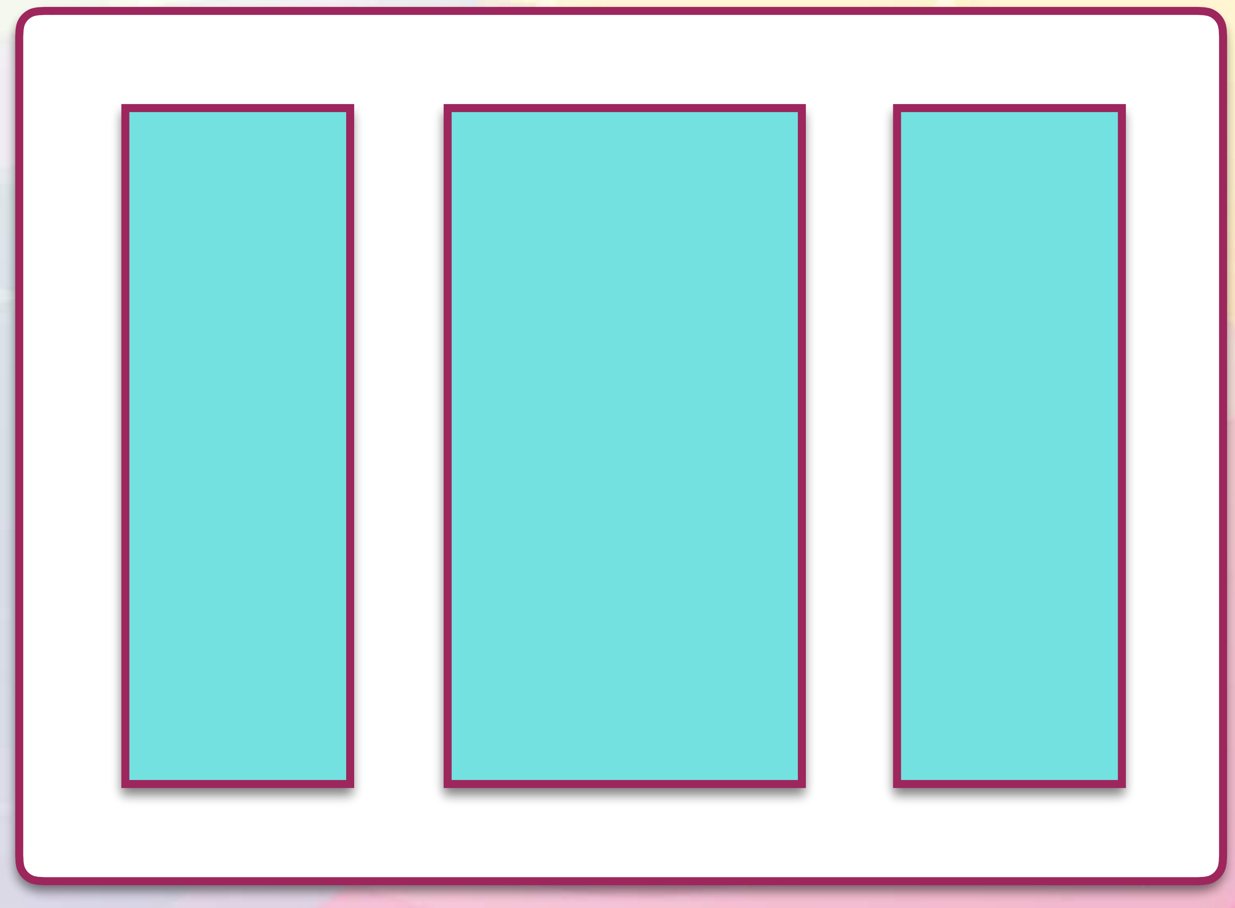
It's best practice to place your app icons in `mipmap` *folders because they are used at resolutions different from the device's current density*

Screencast: Adding an ImageView

The Different Types of Layouts

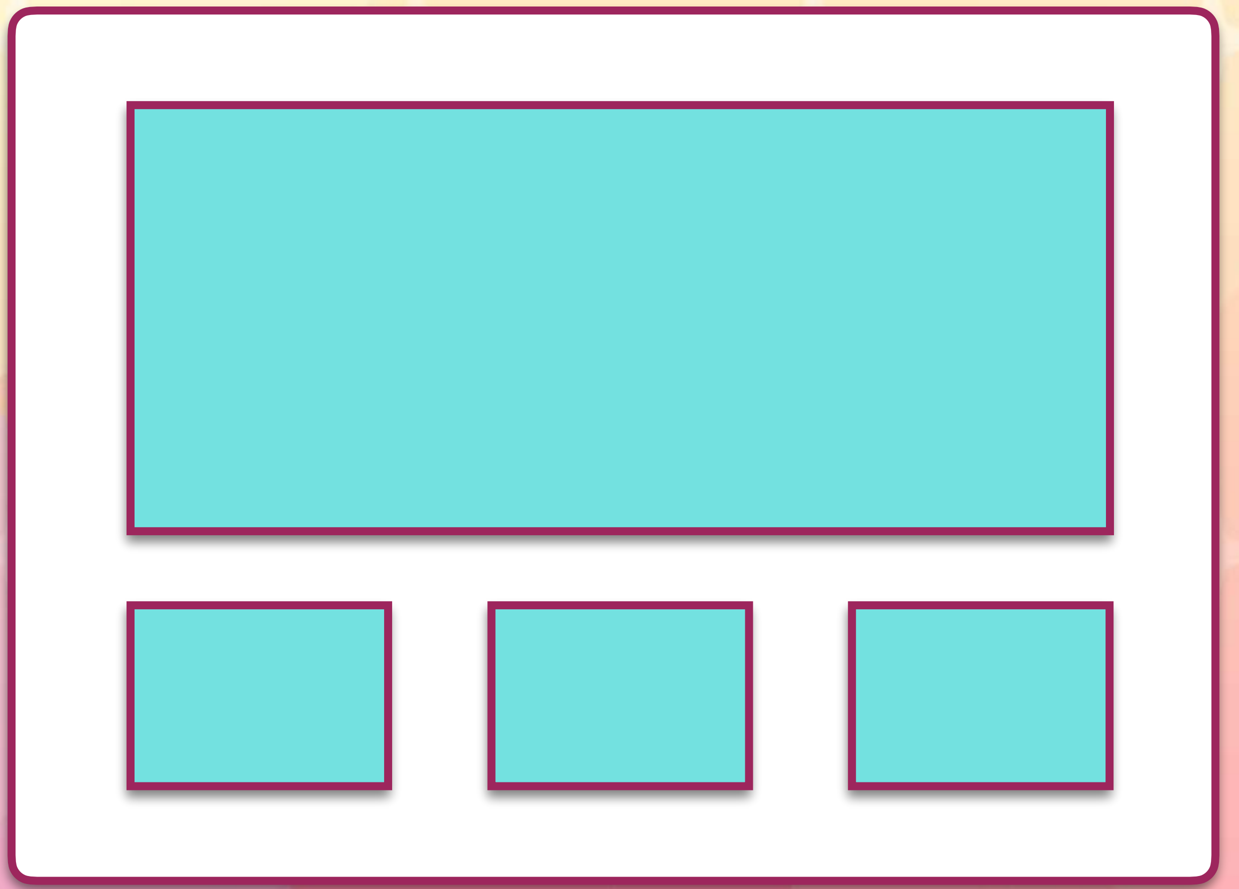
Here are some common layouts built into the Android platform.

LinearLayout



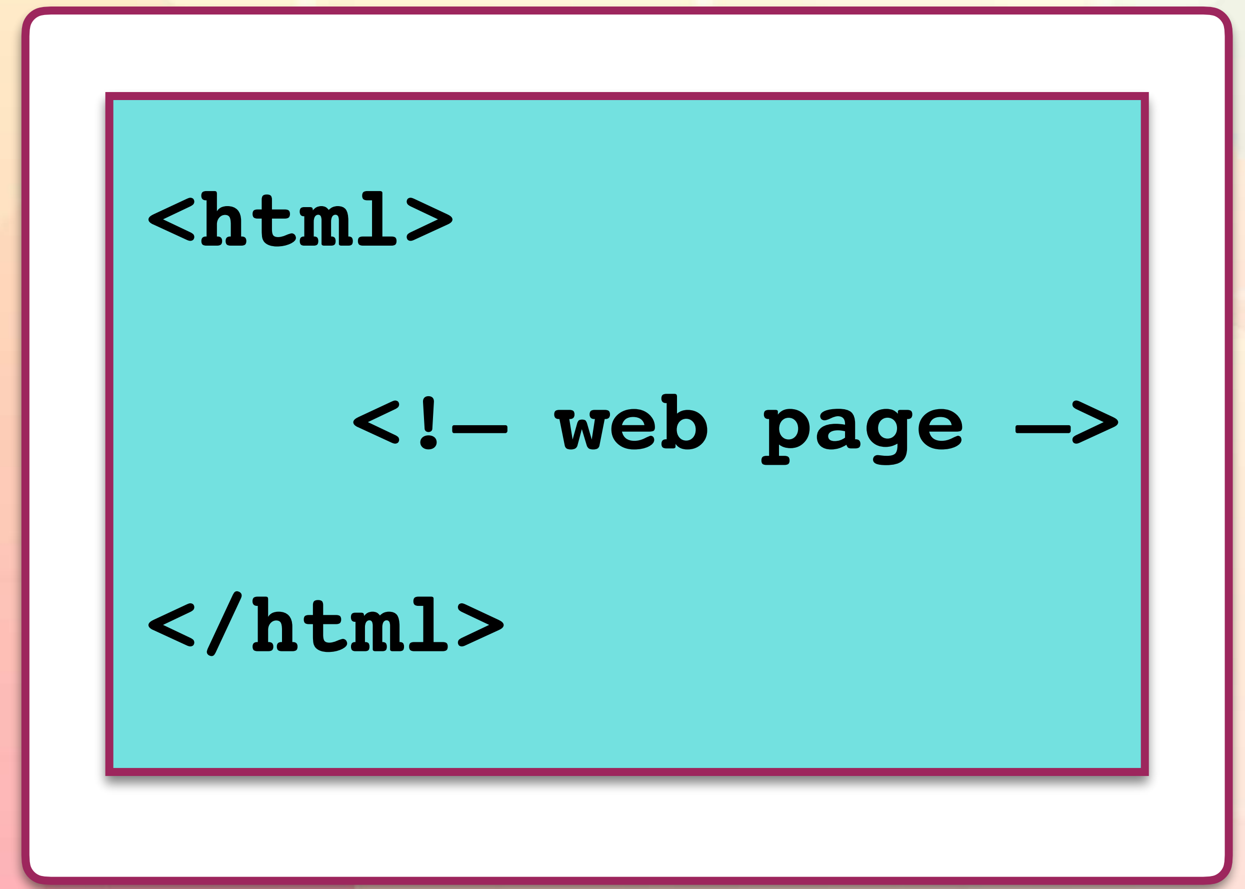
Children organized into a horizontal or vertical row

ConstraintLayout



Child objects are relative to each other (to the left, below, etc.)

Web Layout



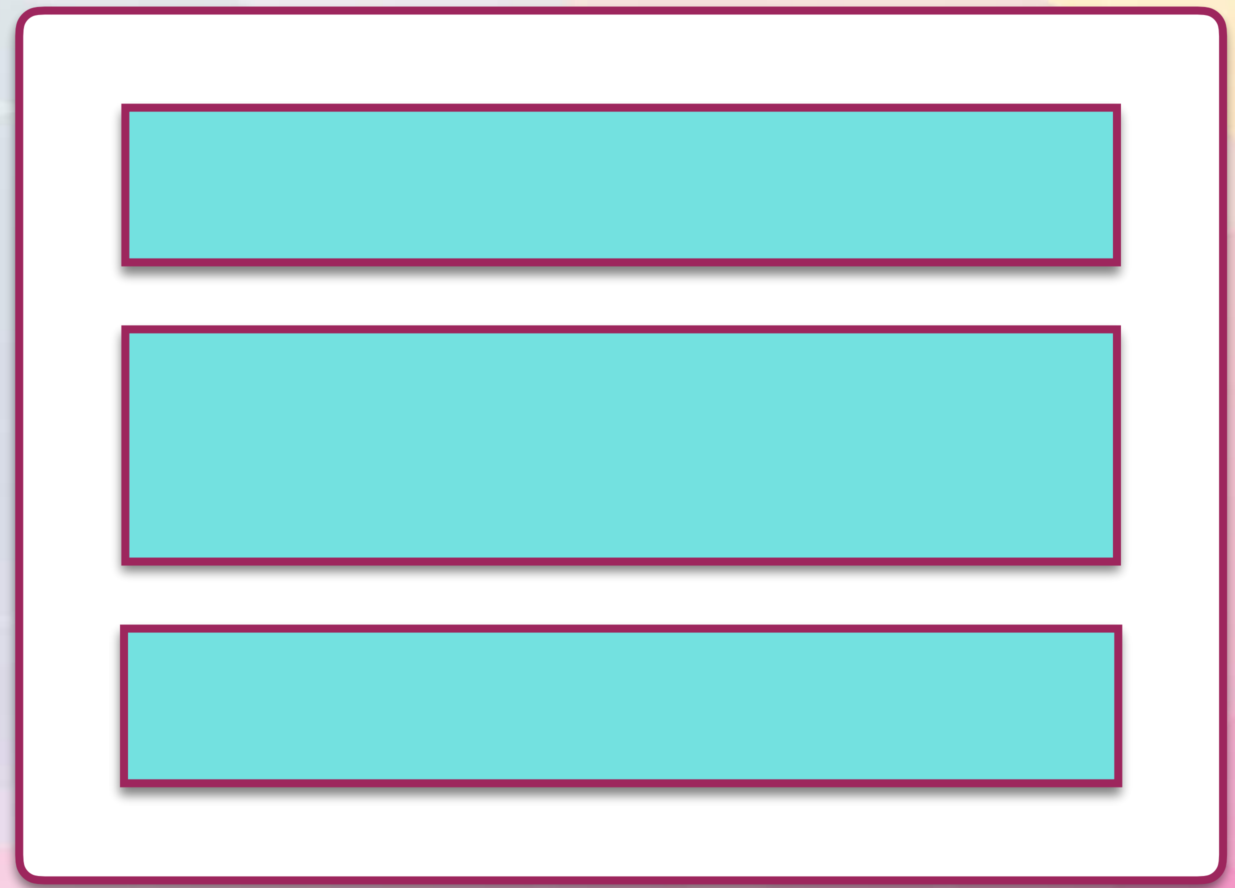
```
<html>
    <!– web page –>
</html>
```

Displays web pages

We Want a Vertical LinearLayout

Since we want our elements displayed one after the other vertically, we'll use a vertical LinearLayout**.**

LinearLayout



Children organized into a horizontal or vertical row

activity_main.xml

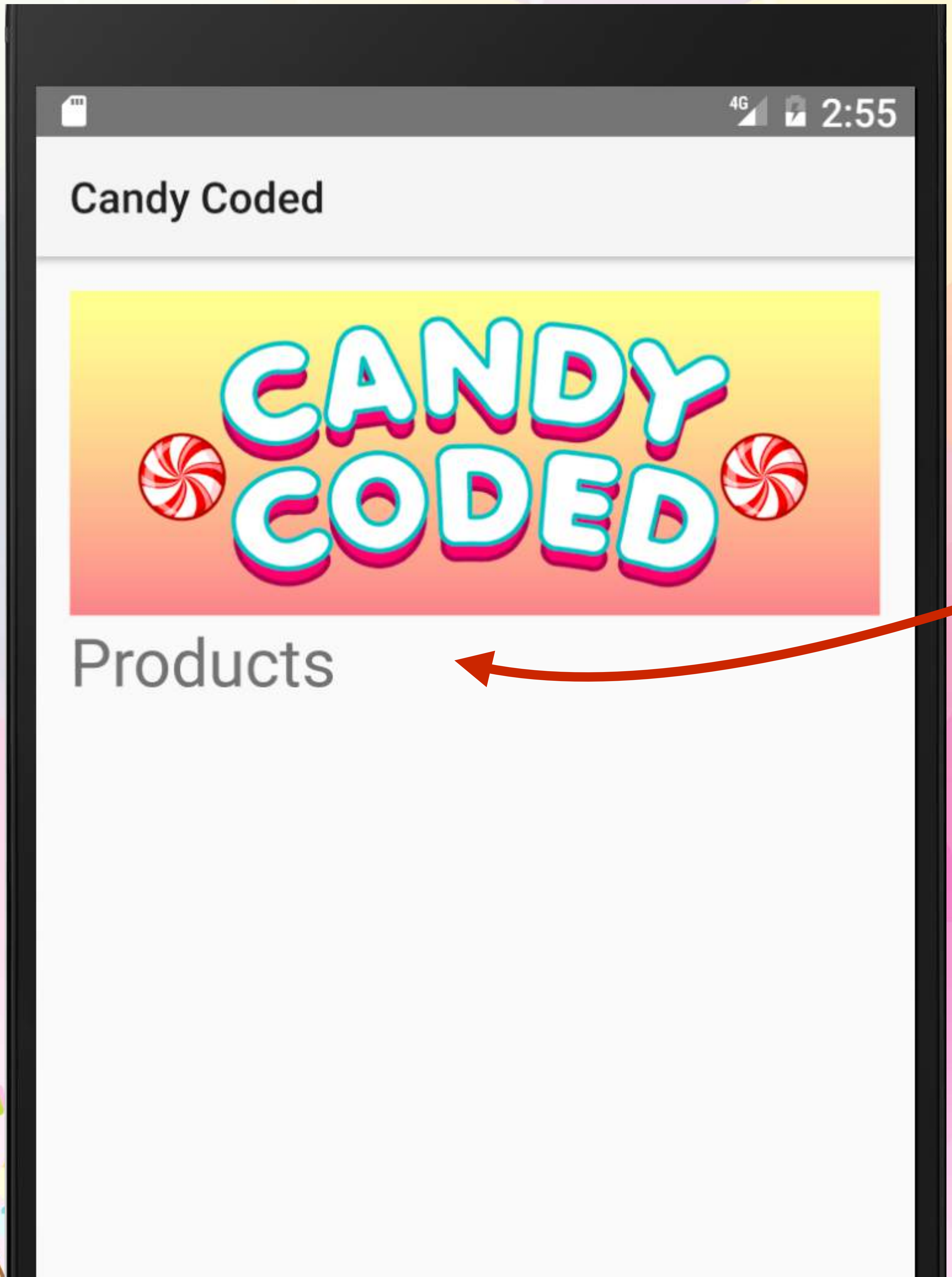
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:orientation="vertical"
    ... />

    <TextView
        ... />

</LinearLayout>
```

We need to replace the default `ConstraintLayout` *with a* `LinearLayout` *with a vertical orientation*

Screencast: Creating a LinearLayout

Demo of the ImageView

Now our app has the ImageView **with the image resource we added.**



We want to center the Products **`TextView`** *and add padding to the top of it, below the* **`ImageView`**.

Screencast: Padding & Layout Gravity

Screencast: Adding Our App Icon

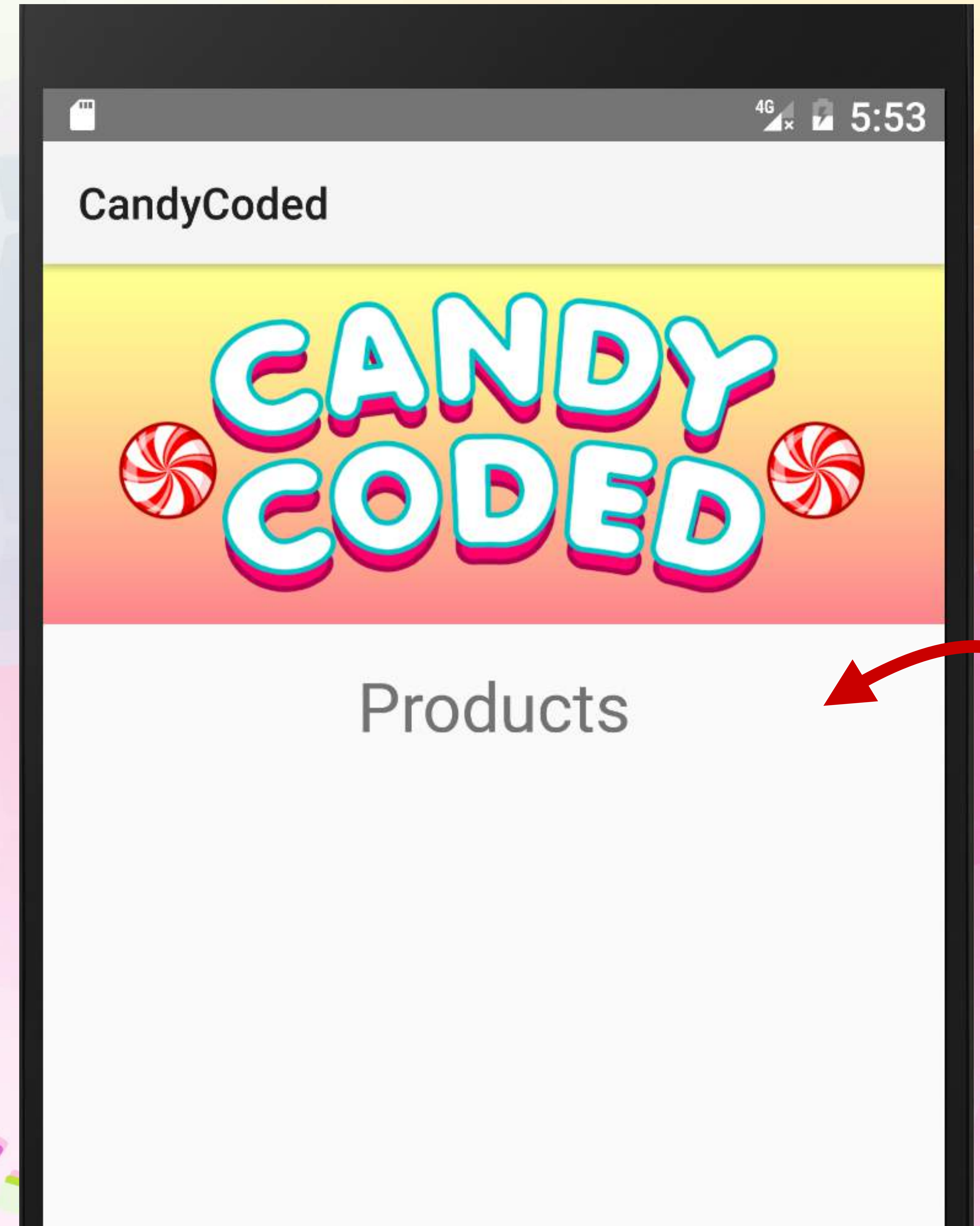
Level 2 – Section 2

Layouts, Views & Images

Updating the Layout With Code

Updating the Layout of a Running App

We set the TextView**'s** text **and the** ImageView**'s source image in** activity_main.xml**, but we can also set it in** MainActivity.java **while the app is running.**



Let's set the `TextView`*'s* `text` *when the app is running in* `MainActivity.java`

Temporary Text in the TextView

In order to see that we've changed the TextView's text in our Java program, we'll make the text set in the layout "Temporary Text" and then change it to "Products" later.

activity_main.xml **XML**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <ImageView .../>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Temporary Text"
        android:id="@+id/text_view_title"/>

</LinearLayout>
```

This was previously:

`"@string/products_title"`

The Starting Code in MainActivity.java

Before we update the TextView's text, let's look at the default code created for us in MainActivity.java.

MainActivity.java **Java**

```java
package com.codeschool.candycoded;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

    }
}
```

Our package

Other packages we need to import

All activities extend from this base class for Activities

The Starting Code in MainActivity.java

Before we update the TextView's text, let's look at the default code created for us in MainActivity.java.

MainActivity.java Java

```
package com.codeschool.candycoded;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

    }
}
```

Gets called when our MainActivity *is starting*

Takes care of all the basic Activity *set up from the base class*

Sets all of the MainActivity*'s content to what's in our* activity_main *layout*

Where to Add Code in MainActivity.java

We'll add our code to the bottom of the onCreate() **method so that our** TextView**'s text is updated when the** Activity **starts.**

MainActivity.java **Java**

```
...

public class MainActivity extends AppCompatActivity {

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



    }
}
```

We'll add all the code to update our `TextView` *here*

Finding a View in Java With findViewById()

First, we need to find our TextView**. To find a** View **in Java code, we can use the** findViewById() **method.**

MainActivity.java **Java**

```
...

    this.findViewById(_____________);

...
```

Find the View within this Activity

The `findViewById` method takes an id as a parameter

❗ *But we never defined an id for our `TextView`!*

Creating an id for the TextView in main_activity.xml

We need an id for the TextView so we can find it in MainActivity.java and then update its text.

activity_main.xml **XML**

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/products_title"
        android:textSize="@dimens/title_size"
        android:id="@+id/text_view_title"/>

<ImageView .../>

</LinearLayout>
```

Now we can find the TextView in our Java code using its id with `R.id.text_view_title`

`R.id` *means the id is defined in the app's resources*

MainActivity.java

```java
...
this.findViewById(R.id.text_view_title);
...
```

Finding a View in Java With findViewById()

findViewById() **returns a generic** View**, which we then need to cast to a** TextView **and then save to a variable.**

MainActivity.java **Java**

We're looking for a TextView*, so we create a* TextView *variable to save our result*

findViewById() *returns a general* View*, so we need to cast it to a specific* TextView

```
...
    TextView textView = (TextView)this.findViewById(R.id.text_view_title);

...
```

Casting the general View *to be a* TextView

Android Studio Automatically Imports TextView

As soon as we type TextView**, Android Studio automatically imports the** TextView **library we need for us by adding this line to the top of our file.**

MainActivity.java **Java**

```java
import android.widget.TextView;


...
    TextView textView = (TextView)this.findViewById(R.id.text_view_title);

...
```

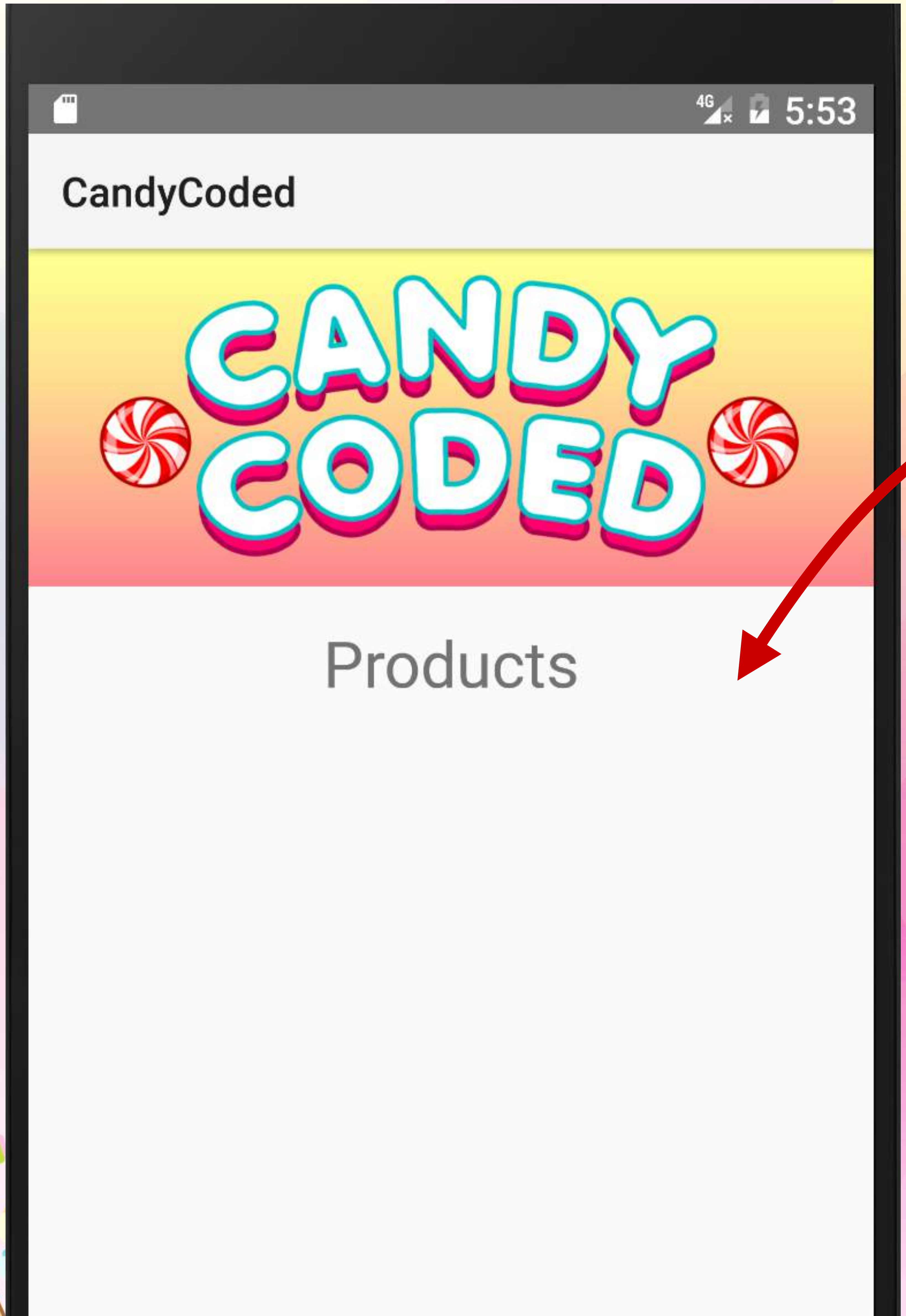
Updating the TextView's Text

Now that we've found our layout's TextView, we can update its text to say "Products".

MainActivity.java **Java**

```java
...
    TextView textView = (TextView)this.findViewById(R.id.text_view_title);

    textView.setText("Products");

...
```

The Title Text Changed When the App Was Running



The `TextView` *changed from* `"Temporary Text"` *to* `"Products"` *once the app started*

We Can Use Our String Resource products_title

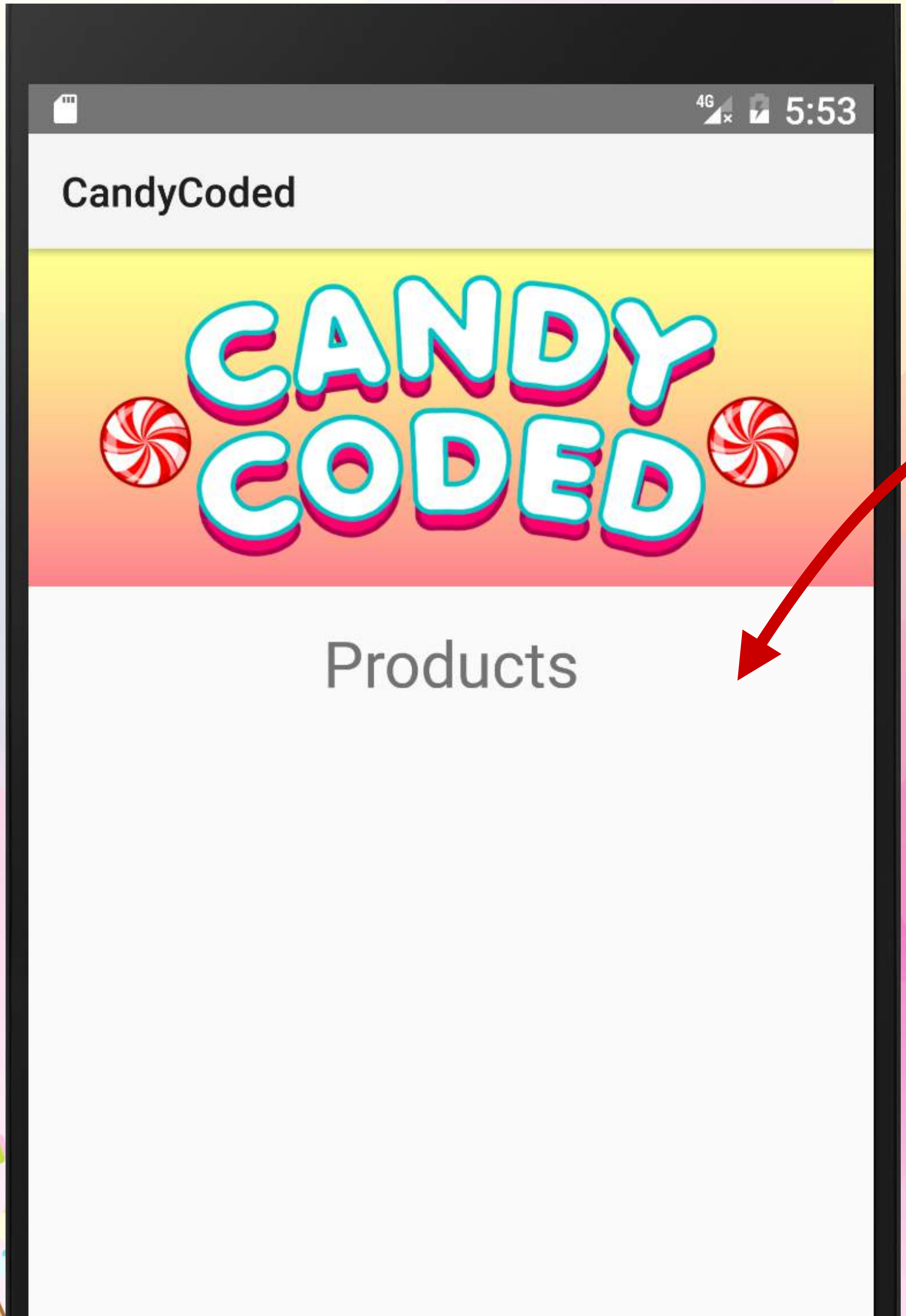
We still don't want to hardcode strings, so we'll use the string resource products_title we defined previously.

MainActivity.java **Java**

```
...
    TextView textView = (TextView)this.findViewById(R.id.text_view_title);
    textView.setText(R.string.products_title);
...
```

We can look up `String` *resource variables with* `R.string.variable_name`

Our App Correctly Displays the products_title



Products

Looking up the `String` *resource* `products_title` *worked because our app still shows* `"Products"`

TRY
ANDROID