
TRY
ANDROID

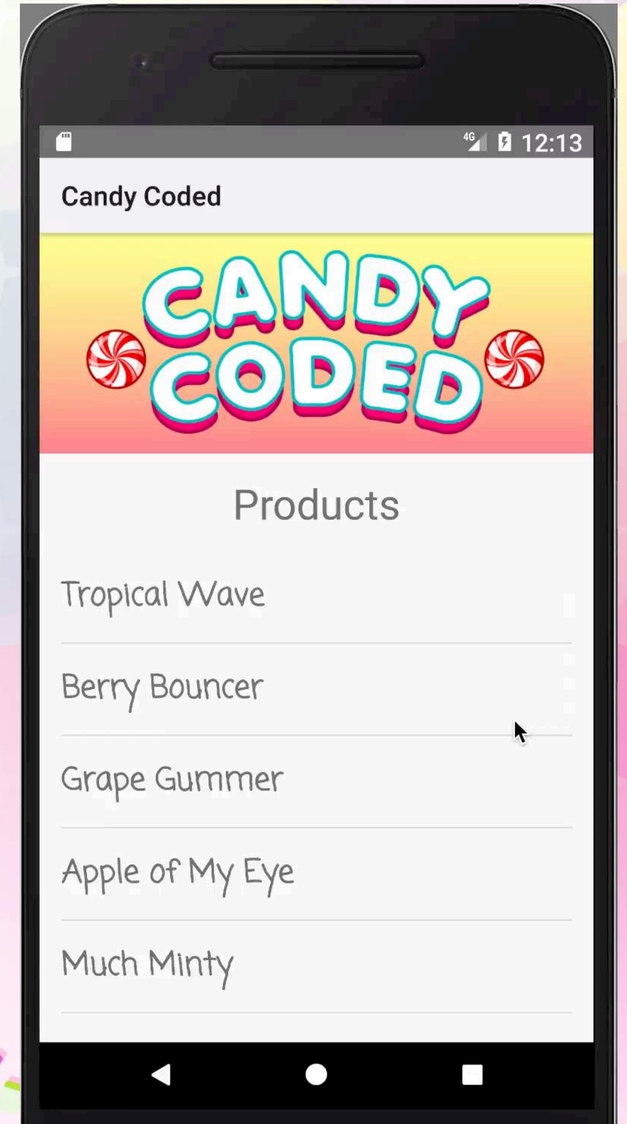
What Is Android?

Android is a mobile operating system developed by Google, primarily for touchscreen smartphones and tablets.

Alpha	Sept 23, 2008	**Ice Cream Sandwich**	Oct 18, 2011
Beta	Feb 9, 2009	**Jelly Bean**	July 9, 2012
Cupcake	April 27, 2009	**Kit Kat**	Oct 31, 2013
Donut	Sept 15, 2009	**Lollipop**	Nov 12, 2014
Eclair	Oct 26, 2009	**Marshmallow**	Oct 5, 2015
Froyo	May 20, 2010	**Nougat**	Aug 22, 2016
Gingerbread	Dec 6, 2010	**Oreo**	Aug 21, 2017
Honeycomb	Feb 22, 2011		

The App We're Going to Make

We're going to make an app to list the candy in our candy store: Candy Coded.

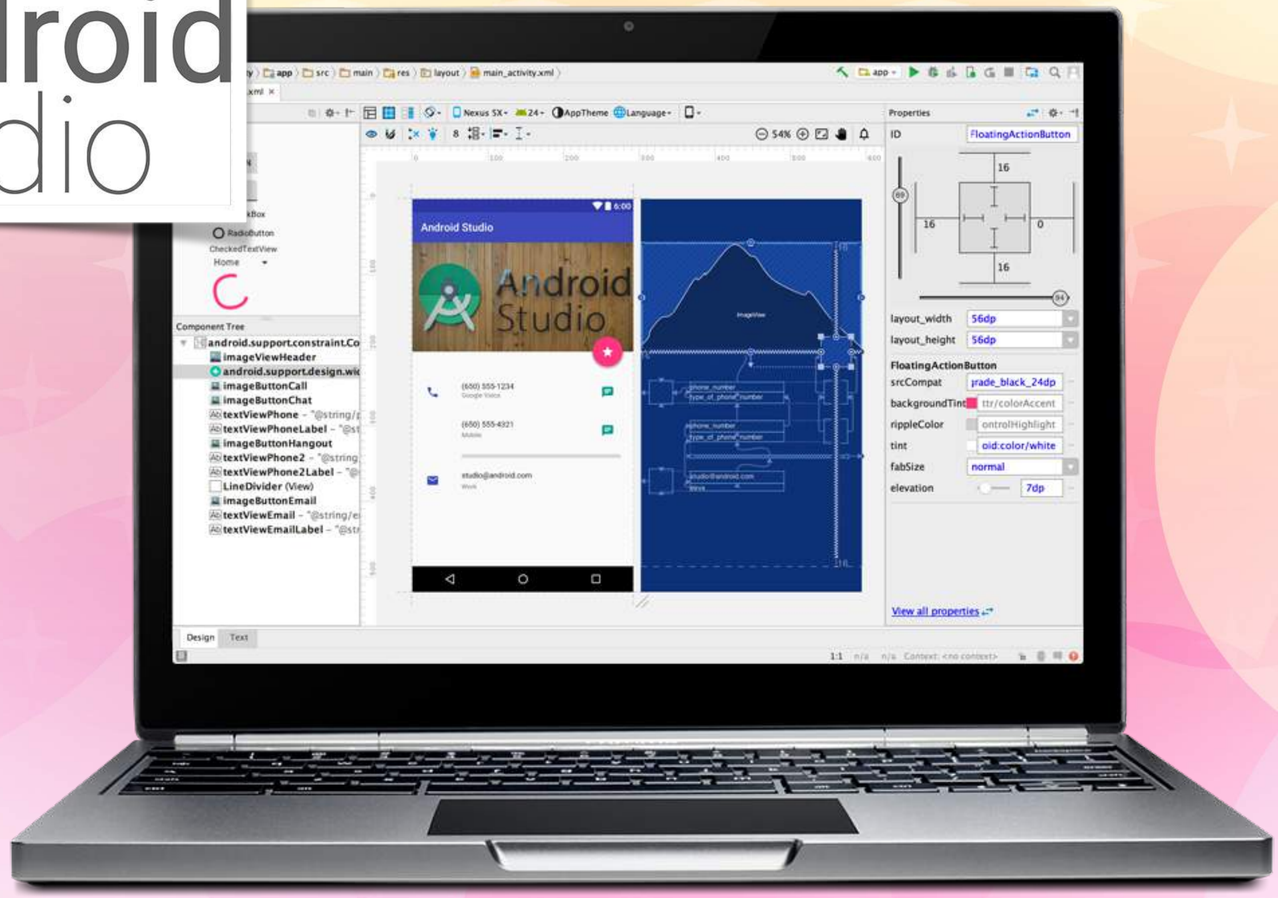


Our app will use `Activity` *and* `View` *classes and* `Layout` *files, which are the main building blocks of an Android app*

TRY ANDROID

Android Studio Helps You Make Android Apps

Android Studio is the official IDE for Android, and you can download it for free here: go.codeschool.com/install-android-studio



TRY ANDROID

Level 1 – Section 1

Cooking Up an App

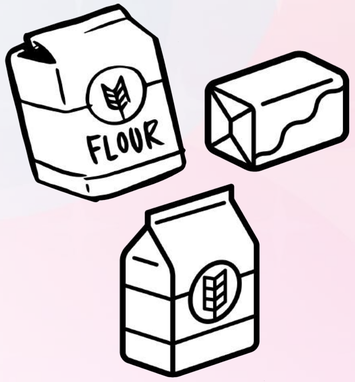
Creating an Android Studio Project

TRY ANDROID

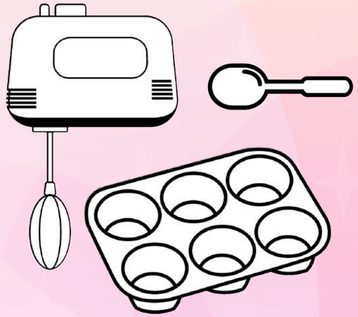
Apps Are Like Cupcakes

What do we need to bake a cupcake?

Ingredients

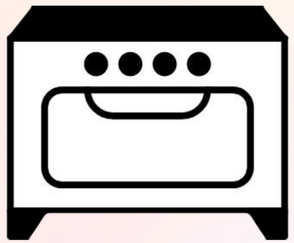


Cooking Utensils



Recipe

Oven

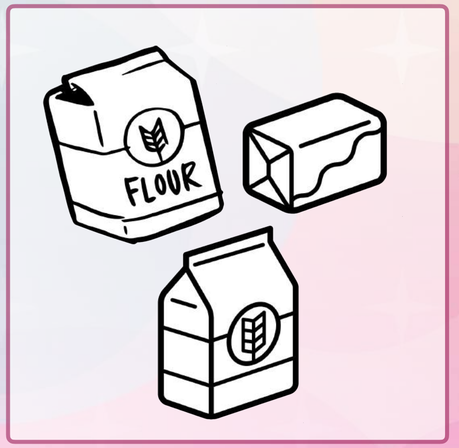


TRY ANDROID

The Android SDK Is Like the Ingredients

The Android SDK provides standard ways to display data in your app, such as activities, layouts, text, images, and buttons.

Ingredients



Layout

Text **Candy Coded**

Images

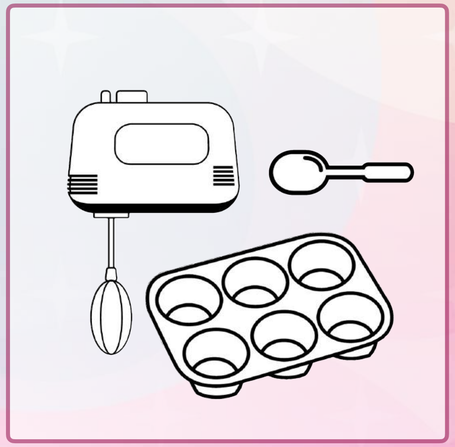
Buttons Add Favorite

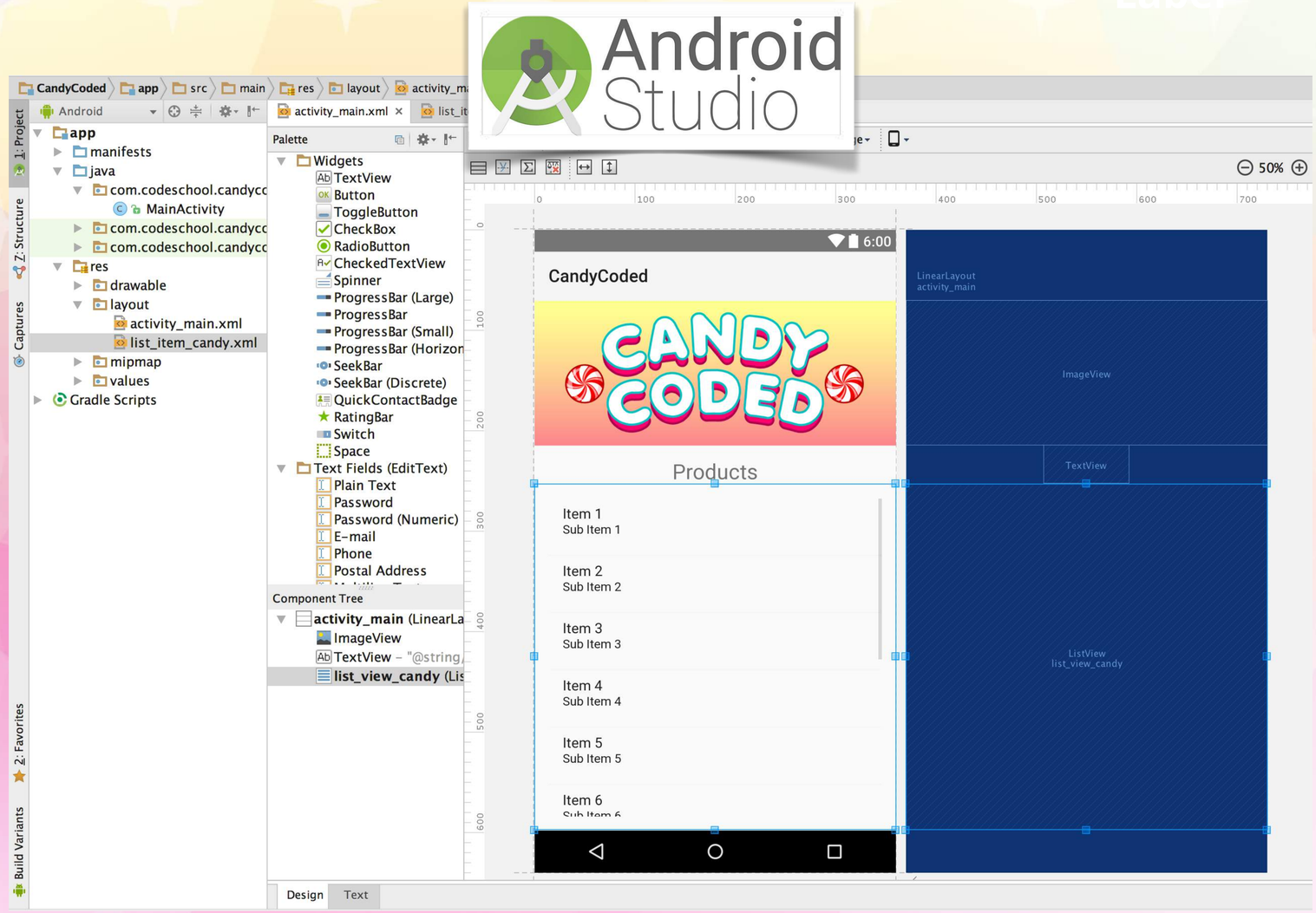
TRY ANDROID

Android Studio Is Like the Kitchen Utensils

Android Studio is a free program provided by Google that you use to write Java code and assemble your app.

Cooking Utensils





Java Is Like the Instructions for Making Cupcakes

You'll write Java code in Android Studio that works with the Android SDK to display your data.

You are the chef!

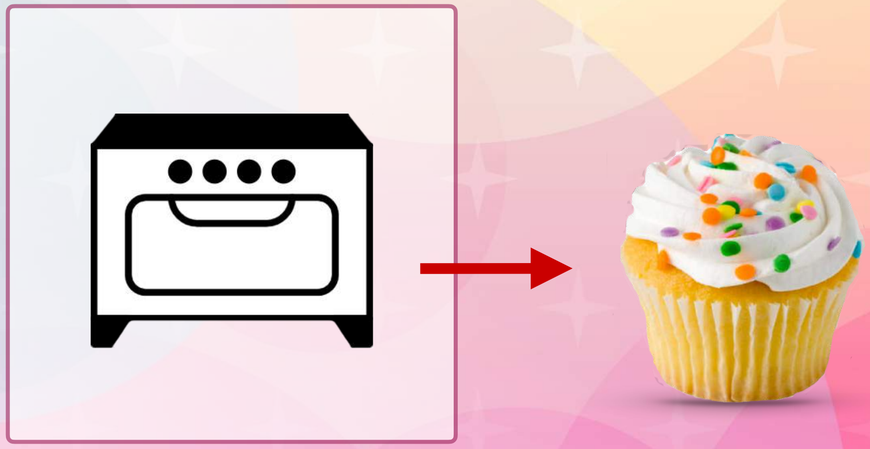
Recipe

TRY ANDROID

Gradle Is Like the Oven

Finally, Gradle is the build system in Android Studio that outputs your final app (or cupcake) into a single APK file.

Oven



Gradle

The final output!

TRY ANDROID

Screencast: Create & Set Up a New Project

TRY ANDROID

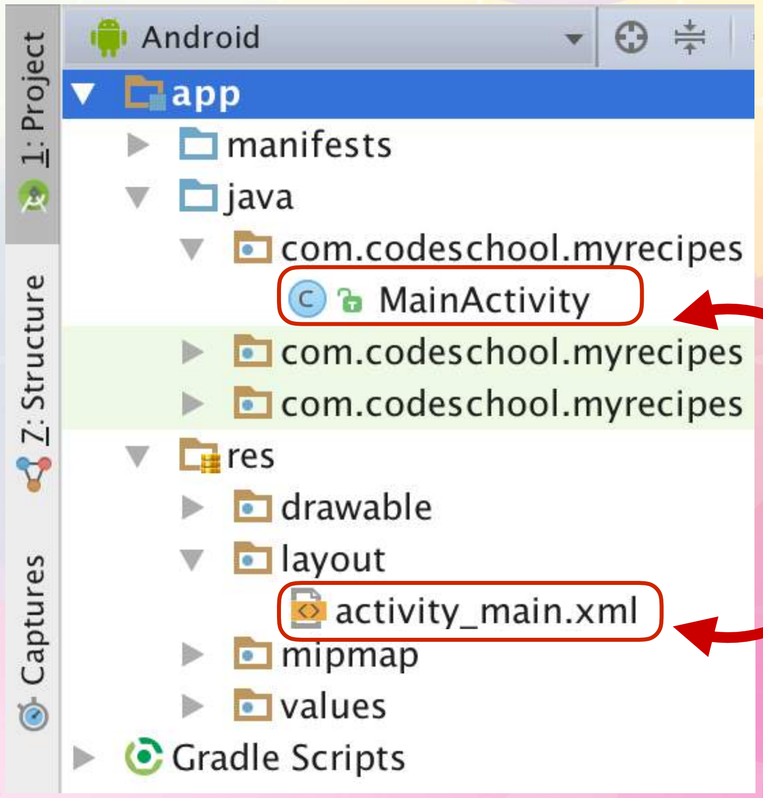
Level 1 – Section 2

Cooking Up an App

Activities, Layouts & Resources

How Is Our App Launched?

The first screen we see in our app is called the MainActivity**, which is launched by the** MainActivity.java **file and displays the** Layout **defined in the** activity_main.xml **file.**

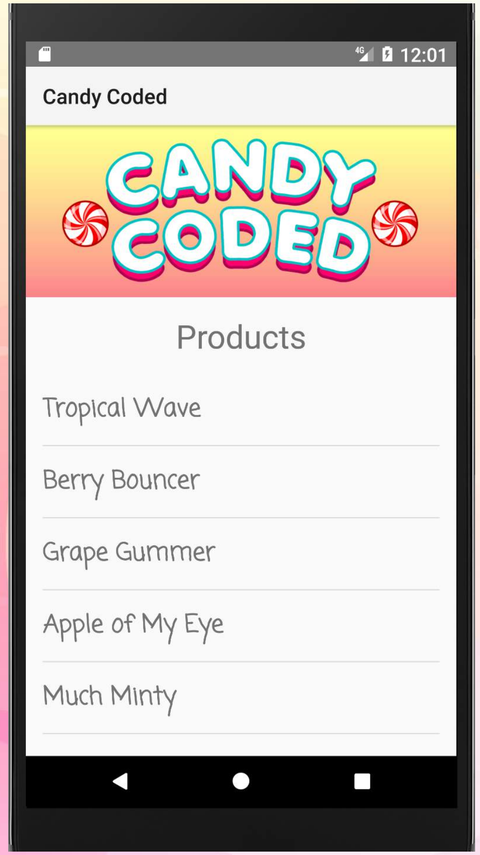


The `MainActivity`

+

The `Layout` *the* `MainActivity` *displays*

=

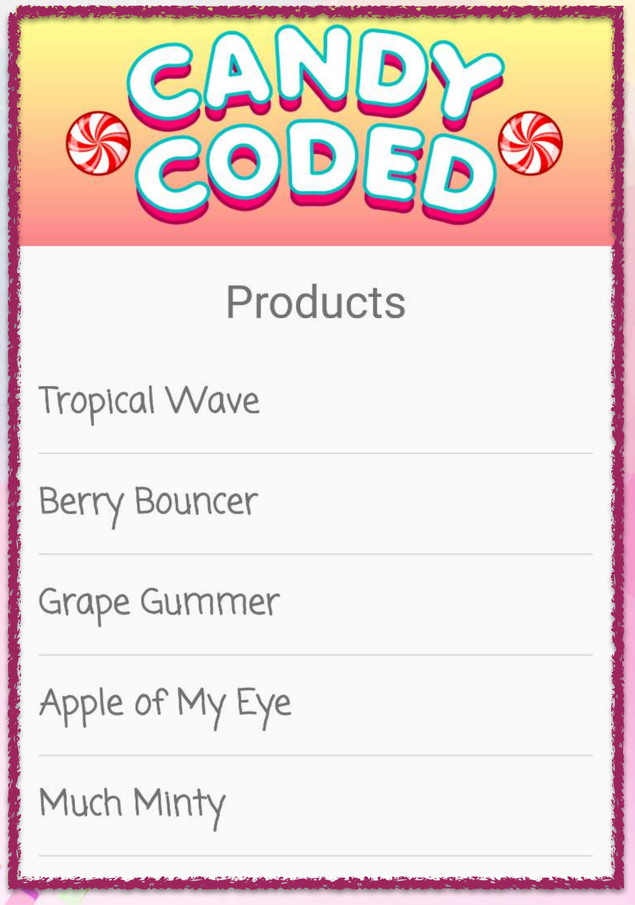


Naming conventions: Java file names use PascalCase, and `Layout` *file names use snake_case where the first word is the type of layout*

Layouts Define the User Interface

A layout describes the visual structure of everything the user will see on that screen.

Layout



A layout can contain elements like text, images, buttons, etc. to display

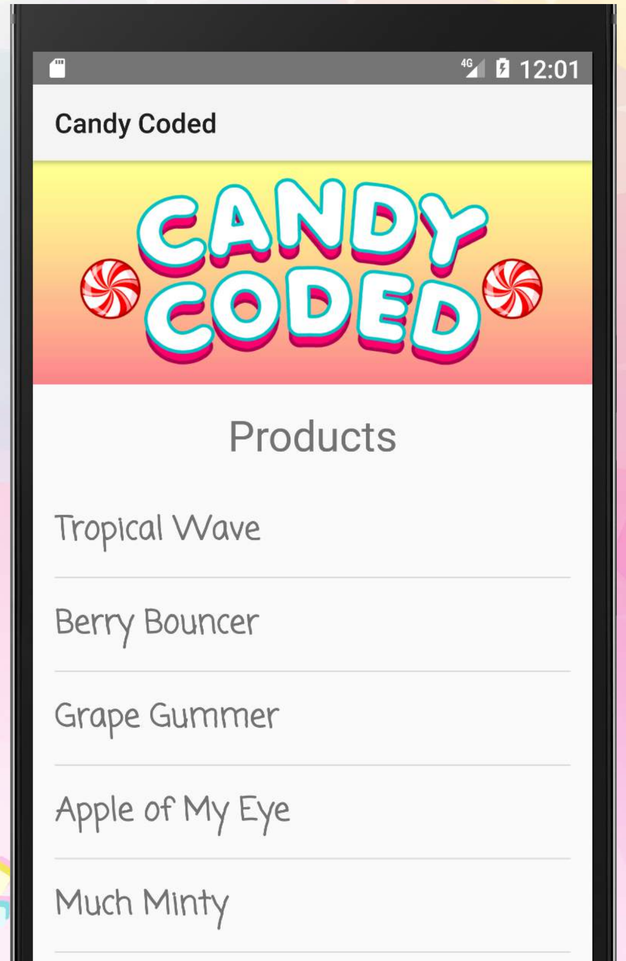
But how do we get them to do something?

TRY ANDROID

An Activity Manages a Layout

An Activity **is a Java file that controls behaviors for our app. An** Activity **usually starts by creating a window to display the** Layout**.**

`MainActivity`



The `Activity` is a Java file that holds behaviors like:

- *Managing layouts*
- *Event handling*
- *Putting data in lists*
- *Opening other activities*

TRY ANDROID

Screencast: Adjusting the Text in the TextView

By default, our Layout **contains a** TextView **with the text “Hello World”. We’ll update it to say “Welcome to Candy Coded!”**

What Dimensions Do We Use?

Android Studio supports the following dimensions: dp**,** sp**,** pt**,** px**,** mm**, and** in**. We'll use only** dp **(for images) and** sp **(for text) in our app.**

dp **- Density-independent Pixels**

For phones with different pixel densities and/or sizes, using `dp` *units (instead of* `px` *units) makes the view dimensions in your layout resize properly*

sp **- Scale-independent Pixels**

`sp` *is the same as* `dp` *but also scaled by the user's font size preferences*

```
android:textSize="24sp"
```

Moto X

300 pixels/in

Google Pixel

500 pixels/in

TRY ANDROID

What Does wrap_content Mean?

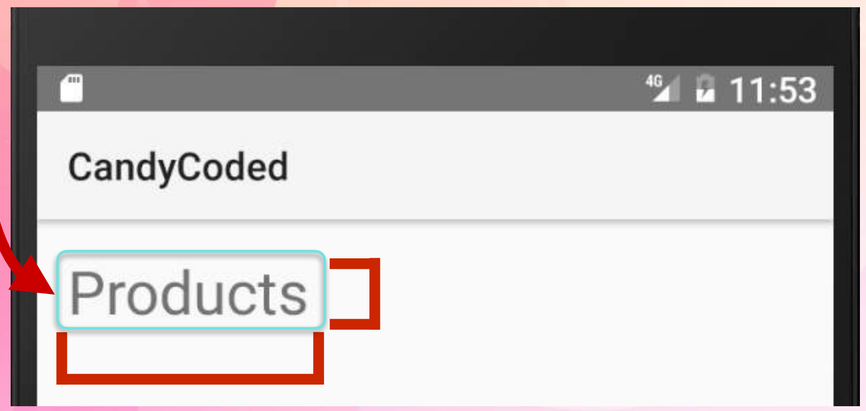
Looking at the TextView **code in detail, we see its** layout_width **and** layout_height **are set to** wrap_content**.**

activity_main.xml

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

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Products"
    android:textSize="24sp" />

</ConstraintLayout>
```

wrap_content means the view will only be large enough to enclose its content

So this TextView will only be wide enough and tall enough to hold the string "Products"



Best Practice: Using the String Resource File

Android Studio warns you if you use hardcoded values, you should use resources instead.

activity_main.xml

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

    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Products"
    ...

</ConstraintLayout>
```

[I18N] Hardcoded string "Products", should use @string resource more...

We want to replace hardcoded strings with a reference to the string resource file:

`"@string/products_title"`

The variable `products_title` *will store the string value* `"Products"`

TRY ANDROID

Screencast: Using Resource Files


TRY
ANDROID

Level 2

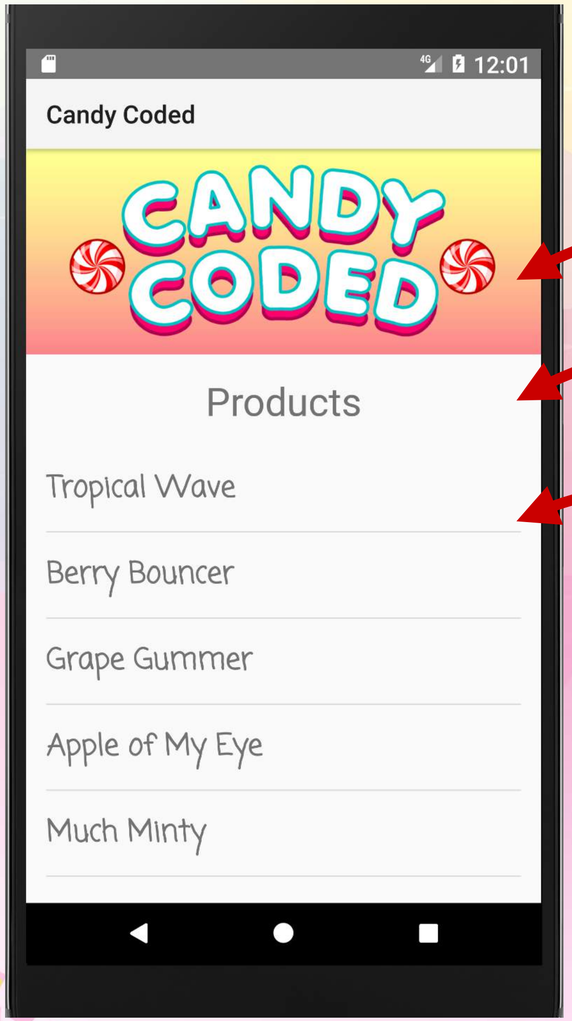
Layouts, Views & Images

Creating a LinearLayout & ImageView

TRY ANDROID

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

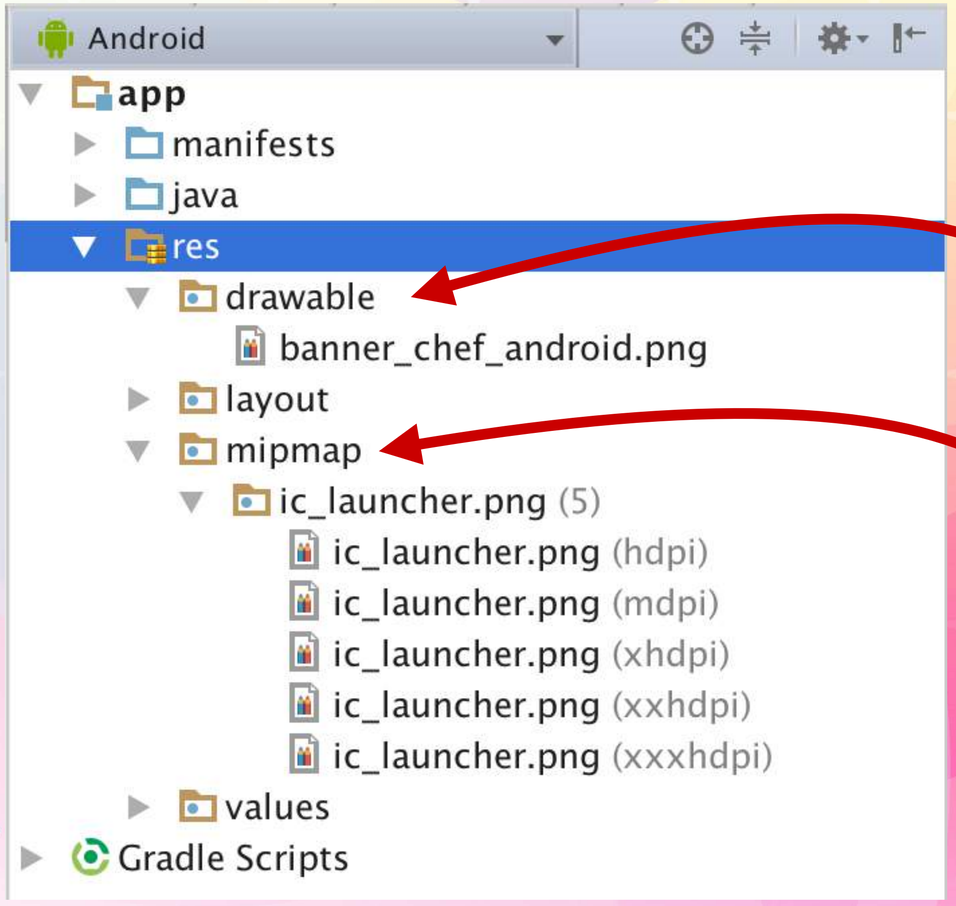
A `TextView` with text "Products"

A `ListView` listing all of our candy products

TRY ANDROID

Android Image Resources

Image resources will either live in the drawable **or** mipmap **folders**

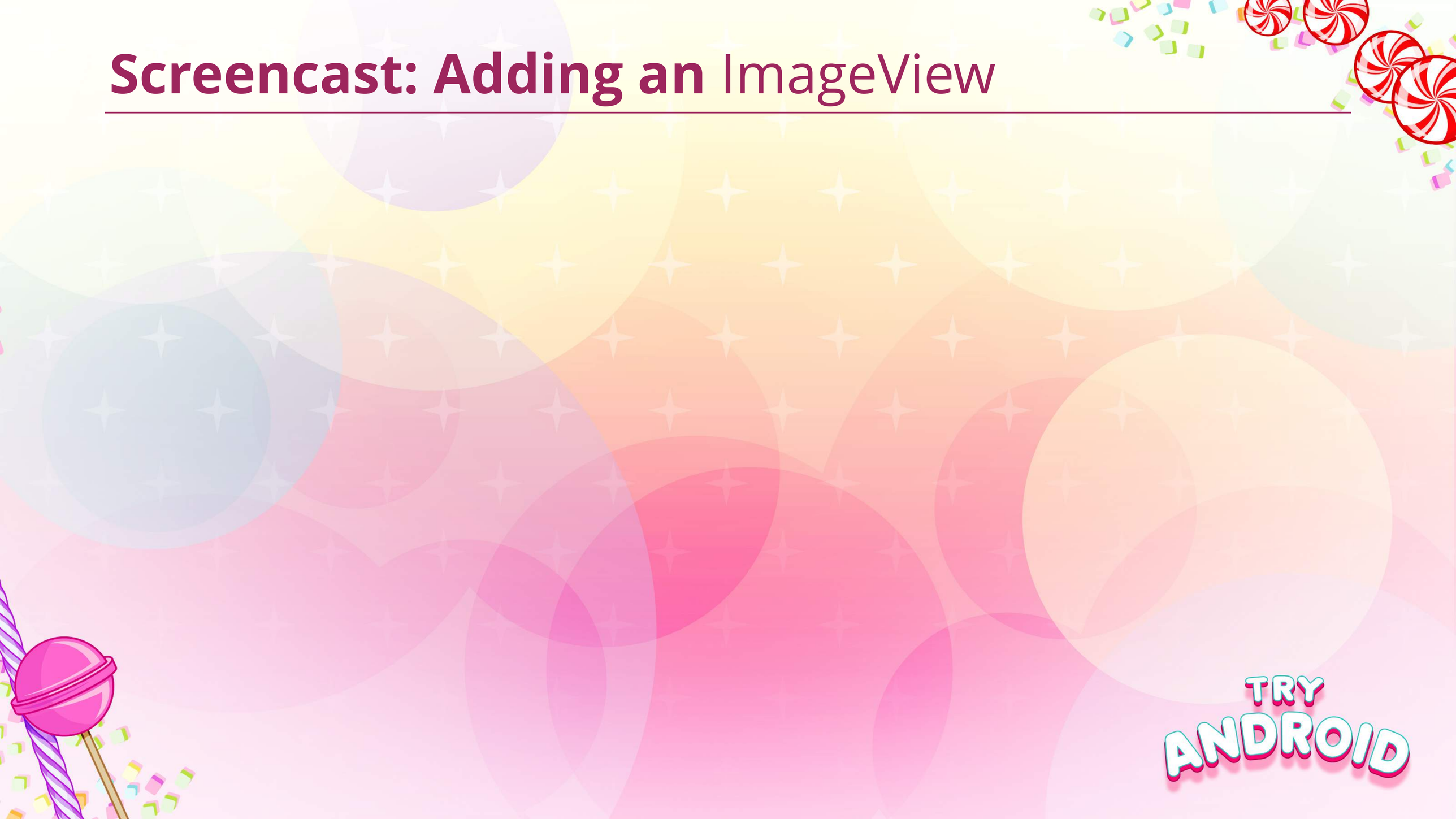


Images that are not affected by changes in scale can be put in the `drawable` *folder under* `res`

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*

TRY ANDROID

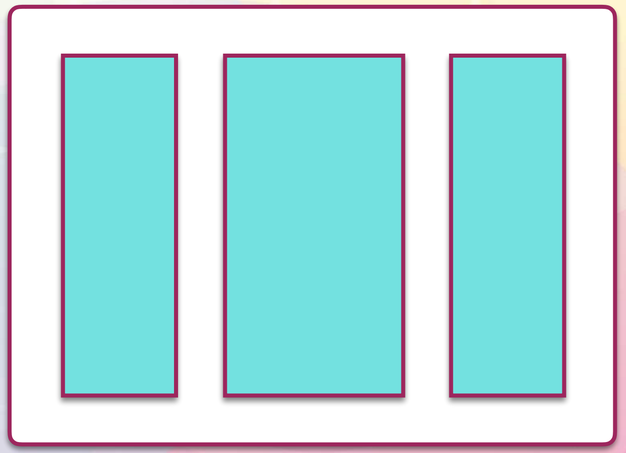
Screencast: Adding an ImageView



The Different Types of Layouts

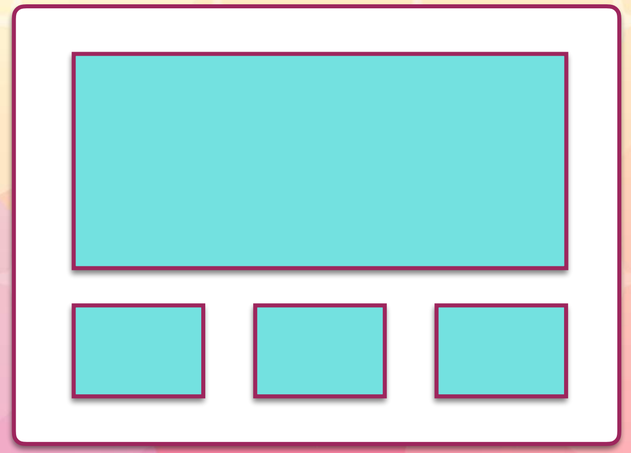
Here are some common layouts built into the Android platform.

LinearLayout



Children organized into a horizontal or vertical row

ConstraintLayout



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

Web Layout

Displays web pages

TRY ANDROID

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

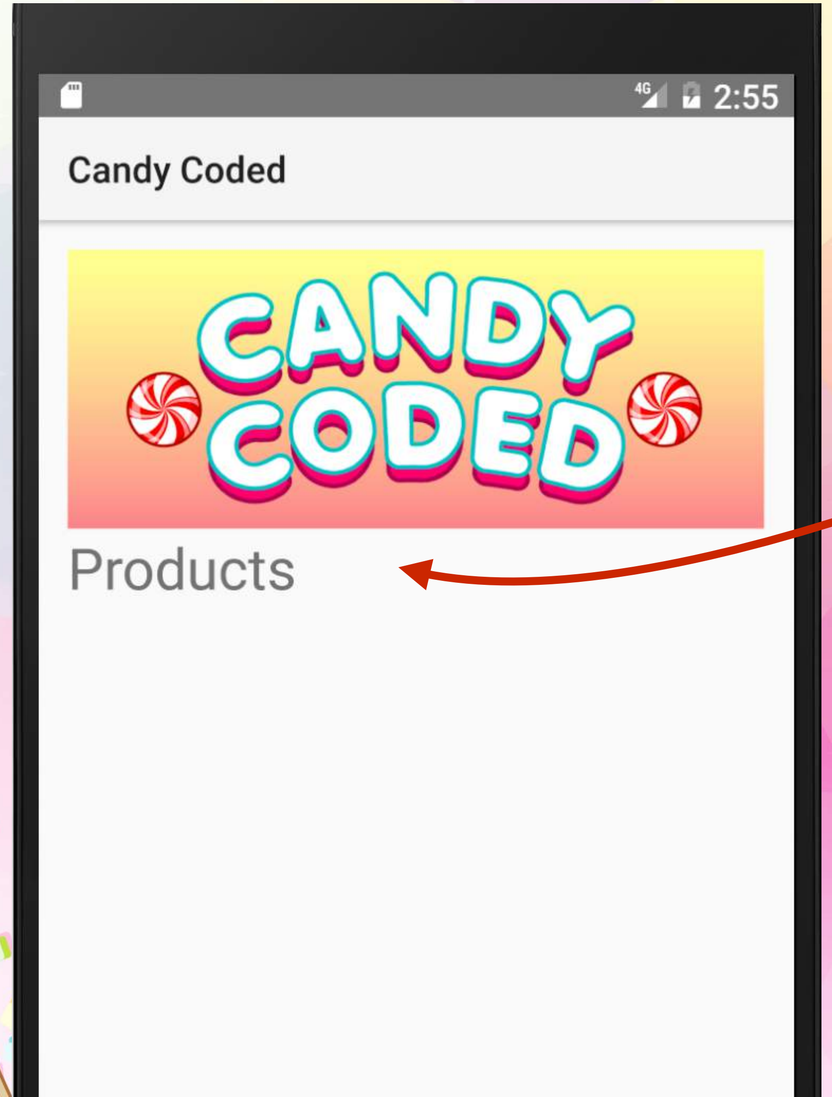
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:orientation="vertical"
    ... />

    <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.**

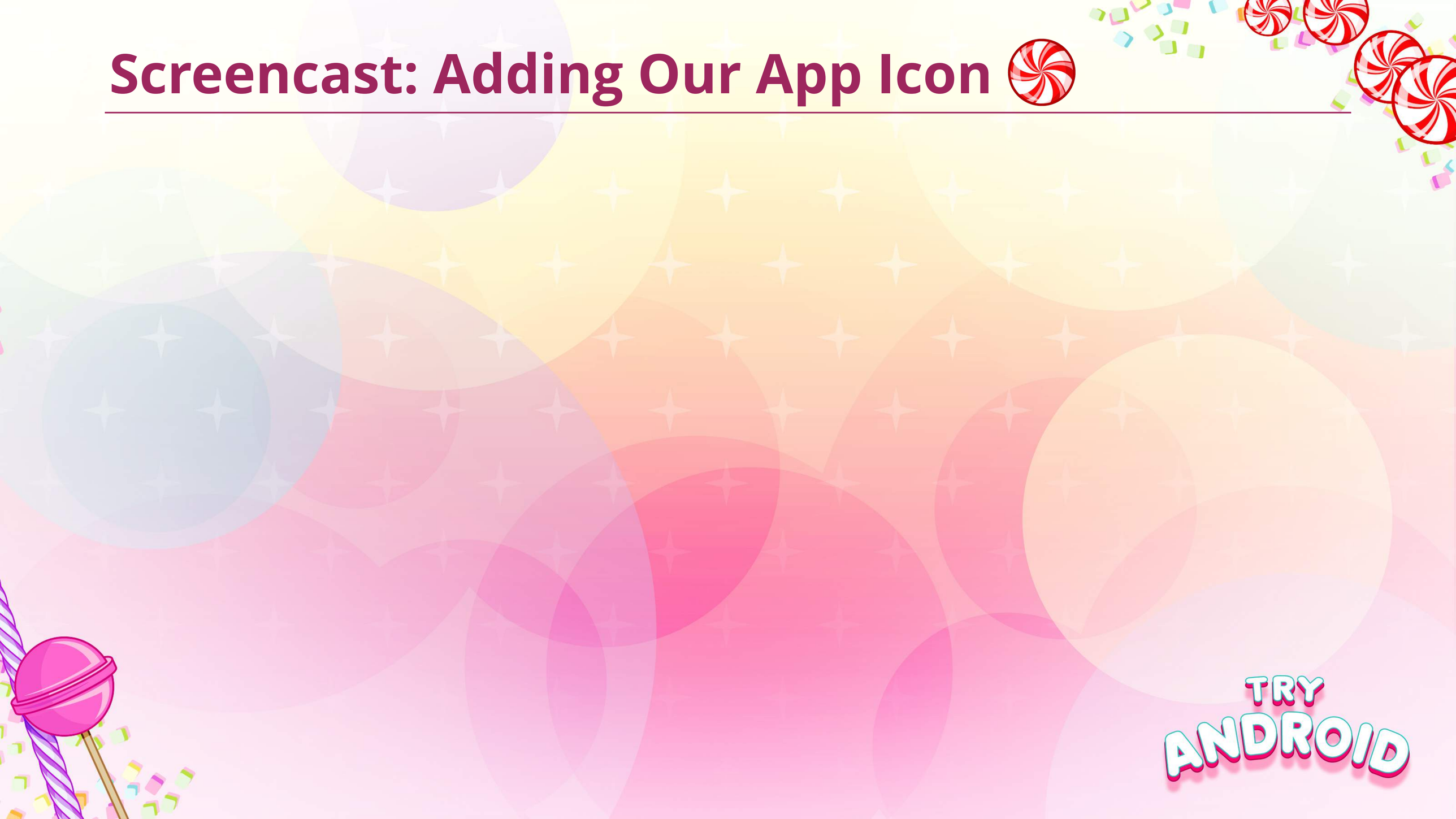


We want to center the Products **`TextView`** *and add padding to the top of it, below the* **`ImageView`**.

TRY ANDROID

Screencast: Padding & Layout Gravity

Screencast: Adding Our App Icon



Level 2 – Section 2

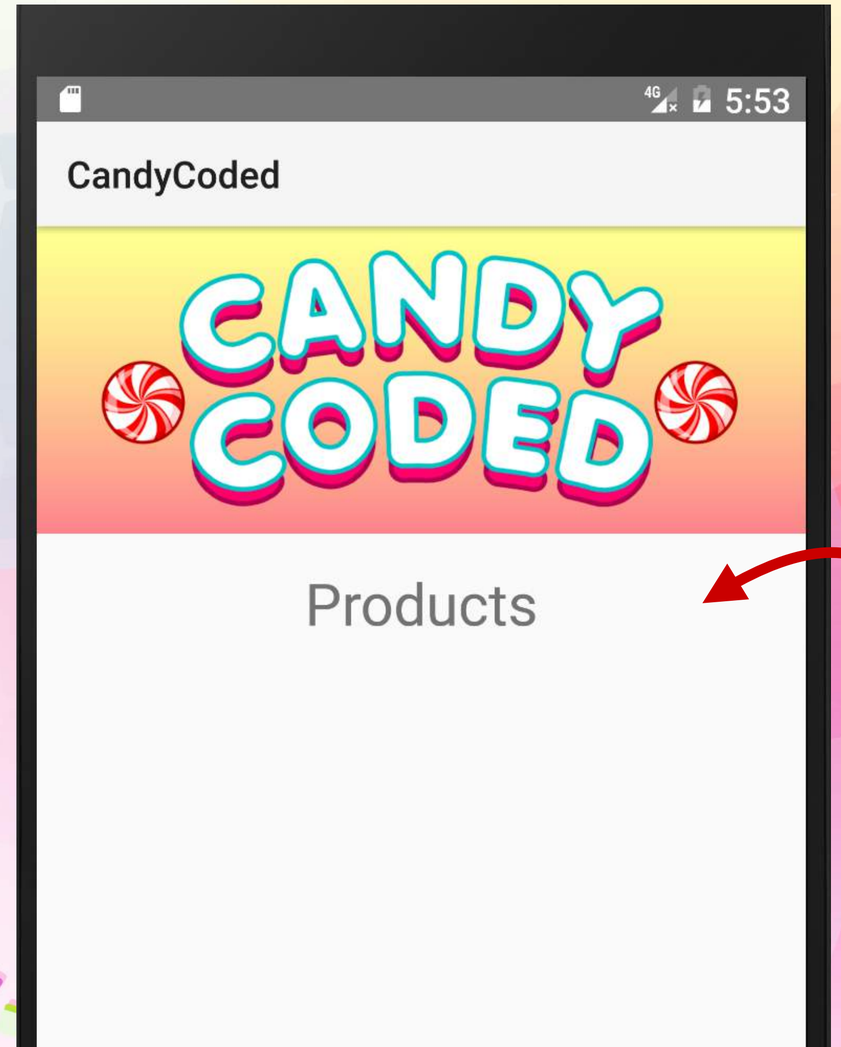
Layouts, Views & Images

Updating the Layout With Code

TRY ANDROID

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`

TRY ANDROID

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"`

TRY ANDROID

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);

    }
}
```

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

```
...
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);

...
```

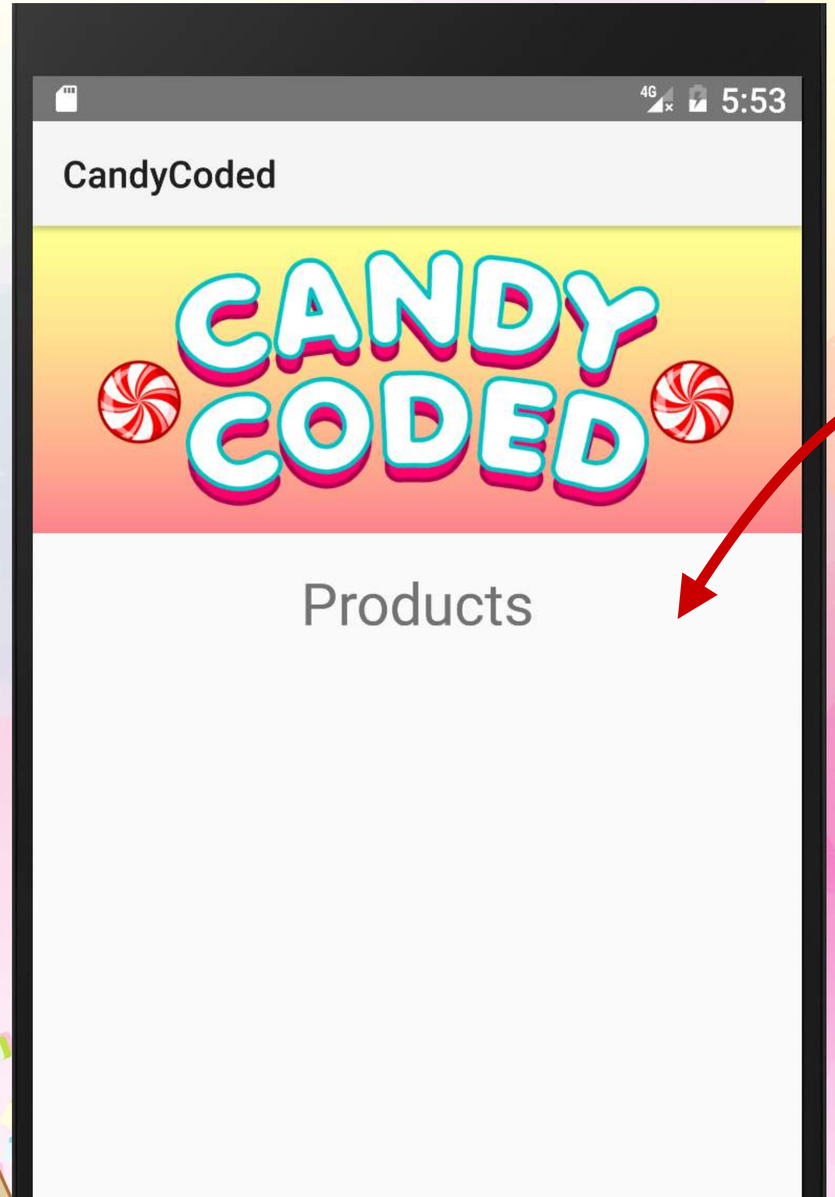
Updating the TextView's Text

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

MainActivity.java **Java**

```
...
    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*

TRY ANDROID

We Can Use Our String Resource products_title

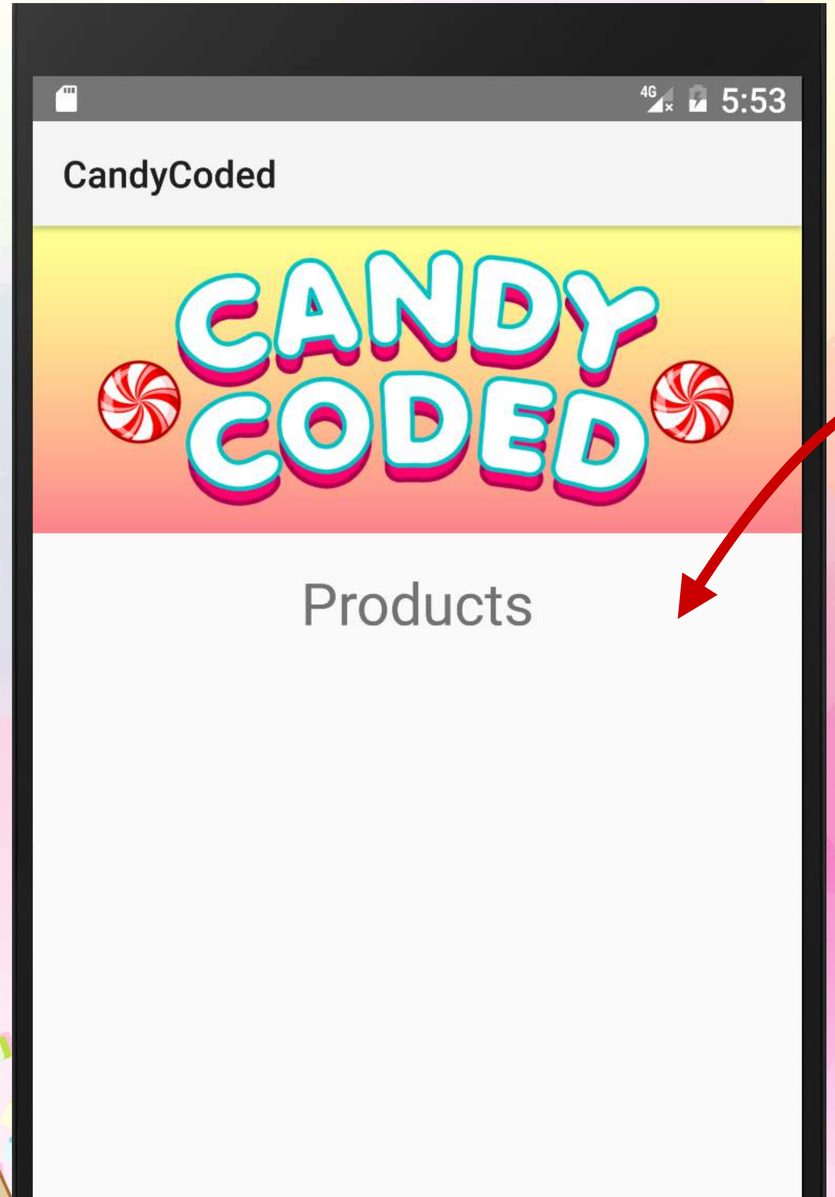
We still don't want to hardcode strings, so we'll use the string resource products_title **we defined previously.**

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



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

TRY ANDROID


TRY
ANDROID

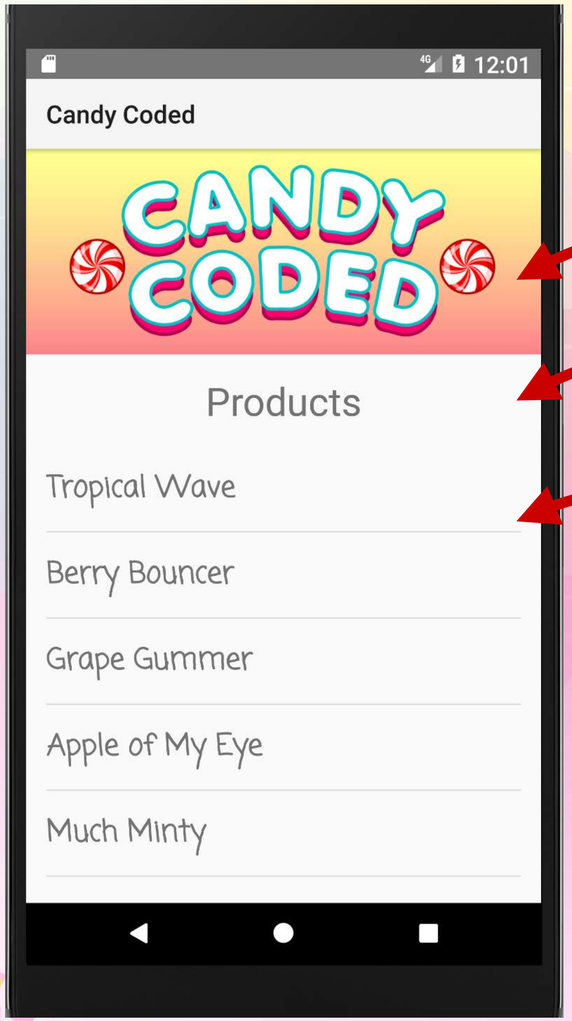
Level 3

ListView & Adapters

Using Adapters to Put Data Into a ListView

Adding a ListView

Now that we've added a TextView **and** ImageView**, we want to add a** ListView **to list our store's available candy.**



An `ImageView` with our Candy Coded logo

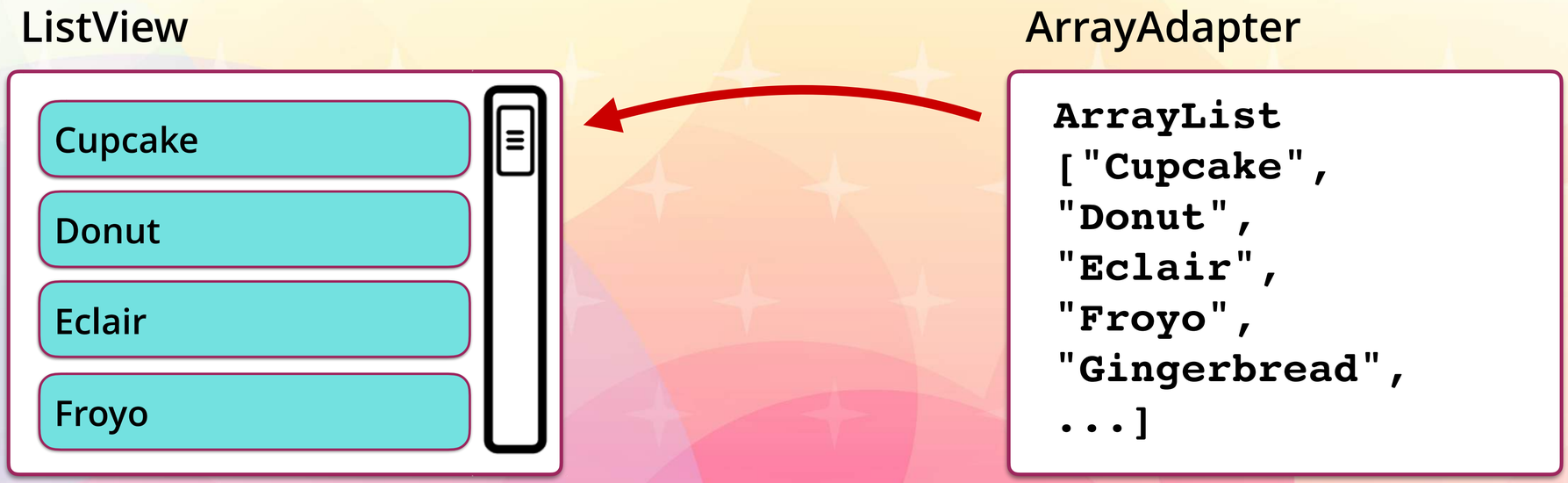
A `TextView` with text "Products"

A `ListView` listing all of our candy products

TRY ANDROID

ListView & ArrayAdapter

A ListView **get its data from a data source via an** Adapter**.**



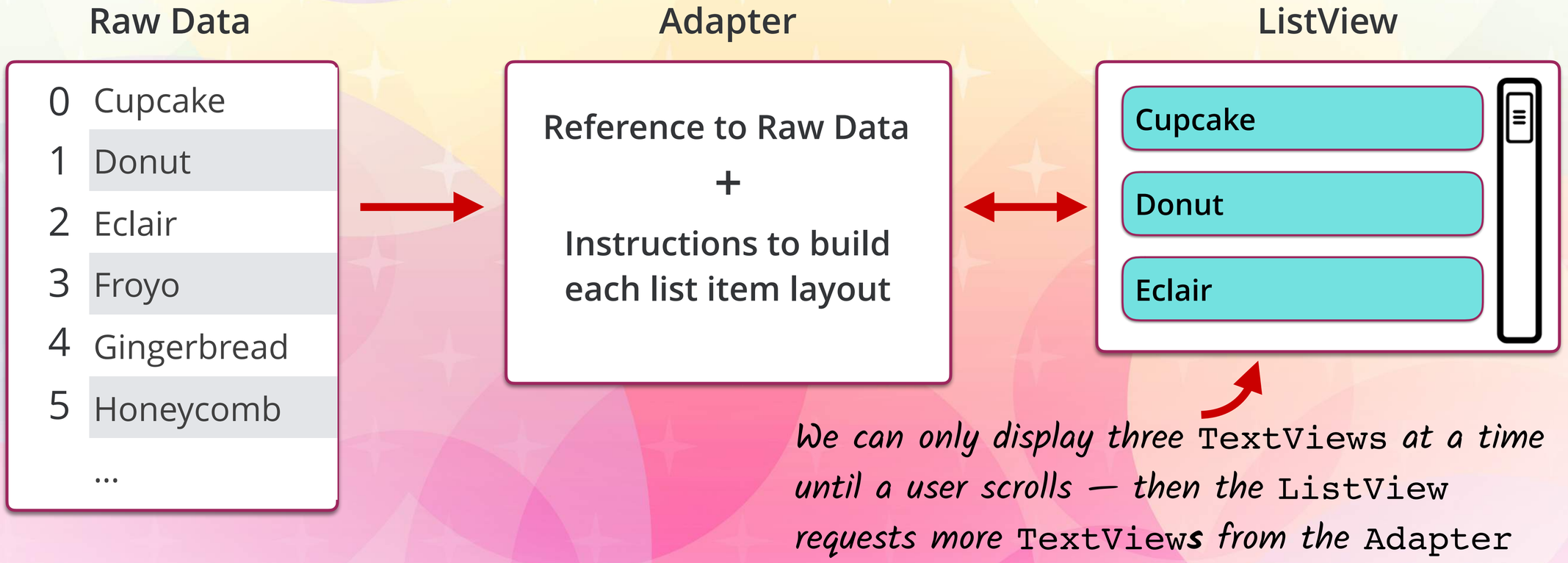
An `ArrayAdapter` *can populate a* `ListView` *with an* `ArrayList`

But why do we need an `Adapter`*? Can't we just send a* `ListView` *a list?*

TRY ANDROID

How Adapters Work

The ArrayAdapter **allows us to only create the items requested by the** ListView**.**



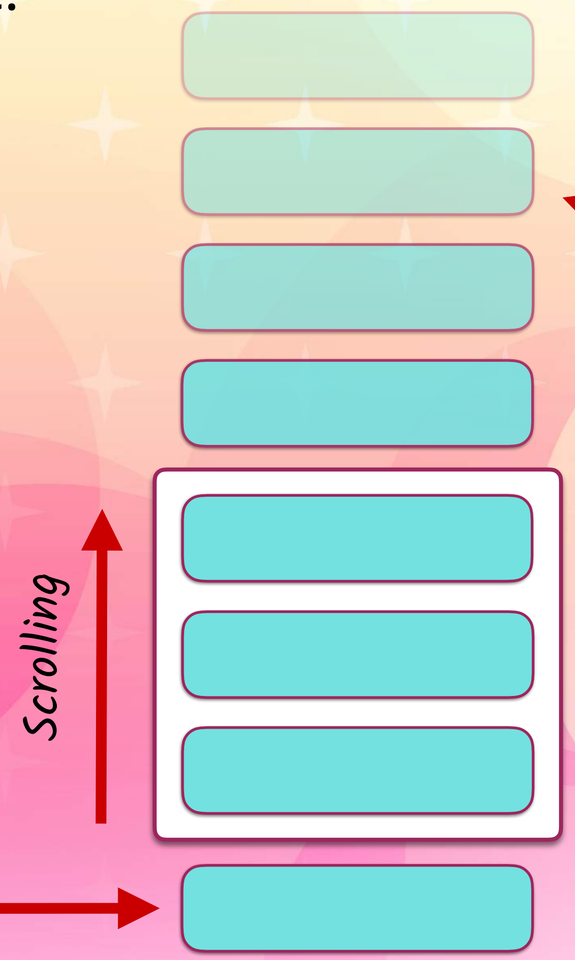
TRY ANDROID

A ListView Without Recycling Views

If we don't recycle the TextViews that make up our ListView cell as they scroll offscreen, we would quickly affect memory and performance.

A `ListView` with its visible `TextViews` and one in either direction:

If we create a new `TextView` as the user scrolls...

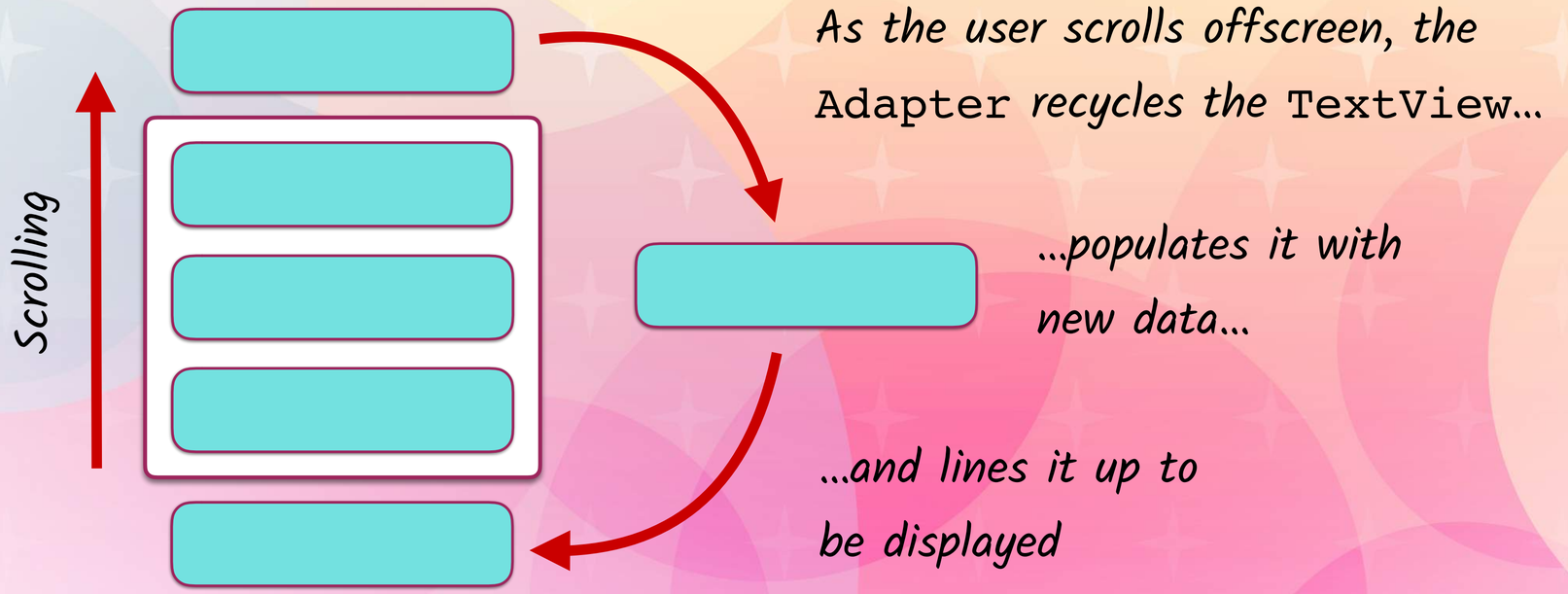


We would quickly affect memory and performance by creating all these cells for a list with thousands of records

TRY ANDROID

How ListView Recycling Works

ListView cells are recycled as they scroll offscreen, and any new cells are just recycled cells filled with new data.



As the user scrolls offscreen, the `Adapter` *recycles the* `TextView`*...*

...populates it with new data...

...and lines it up to be displayed

TRY ANDROID

The Steps to Create the ListView and ArrayAdapter

In order to get our ListView **populated with data from the** ArrayAdapter**, we need to do the following steps:**

Steps to set up a `ListView` *and* `ArrayAdapter`*:*

Layout steps

1) Create a **`ListView`** in the **`main_activity.xml`** layout
2) Create a separate layout to describe what each list item will look like

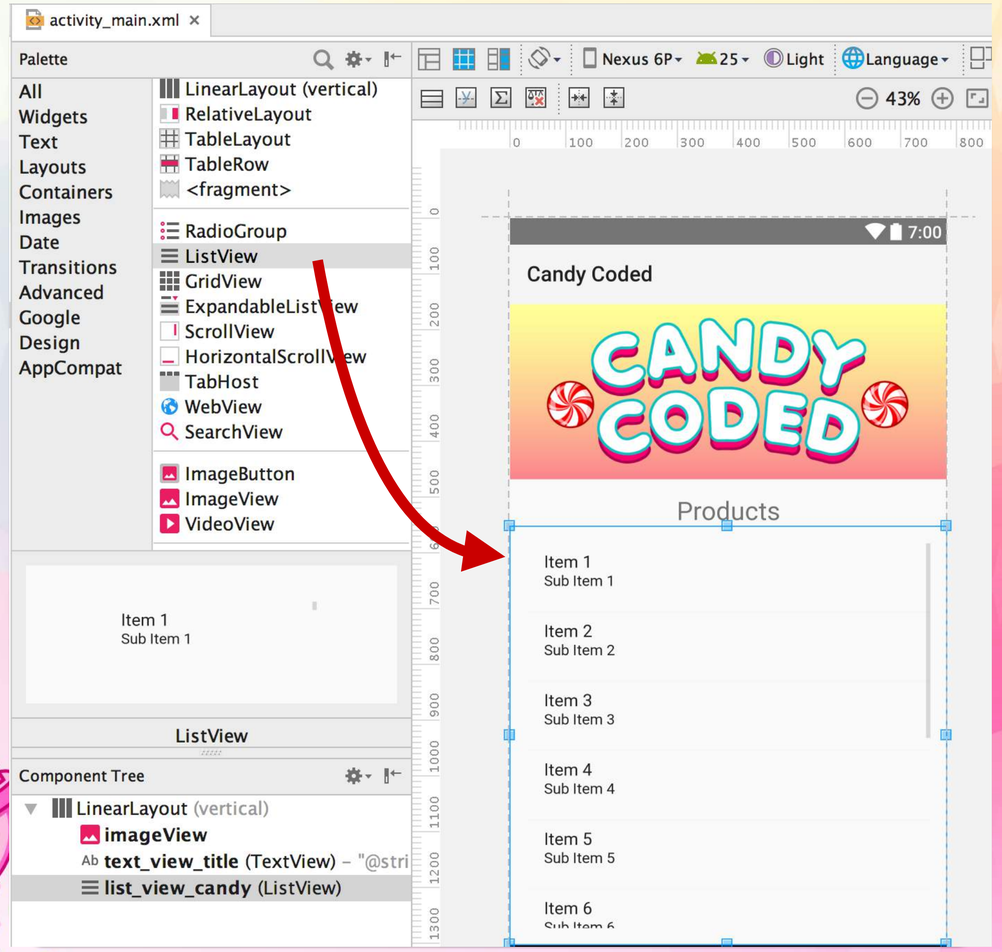
Java steps

3) Get the raw data we want to display into an **`ArrayList`**
4) Create the **`ArrayAdapter`**
5) Connect the **`ArrayAdapter`** to the **`ListView`**

TRY ANDROID

1 Creating a ListView in the MainActivity Layout

To add a ListView **to our** activity_main **layout, we can drag it over in the design view, which will generate the following** ListView **XML.**



activity_main.xml

```
<?xml ...>
<LinearLayout ...>
    <ImageView .../>
    <TextView .../>

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list_view_candy"/>

</LinearLayout>
```

Our ListView's Properties

The ListView**'s** layout_width**,** layout_height**, and** id **were set to default values for us.**

activity_main.xml

```
<?xml ...>
<LinearLayout ...>
    <ImageView .../>
    <TextView .../>

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list_view_candy"/>

</LinearLayout>
```

`match_parent` *means the view will try to be as big as its parent by expanding to take up the remaining space in the layout*

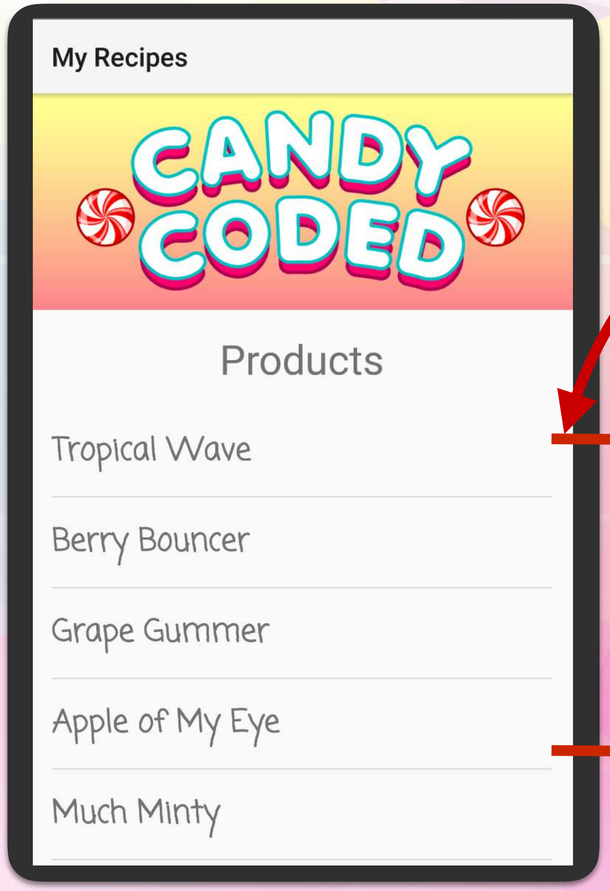
`@+id` *means this id will be created as a resource at runtime*

`list_view_candy` *is this* `ListView`*'s id we picked, but we could name it anything*

TRY ANDROID

What Does match_parent Mean?

The ListView**'s** layout_width **and** layout_height **are** match_parent**.**



`match_parent` *as the* `layout_height` *means the list will take up the available height left in its parent layout*

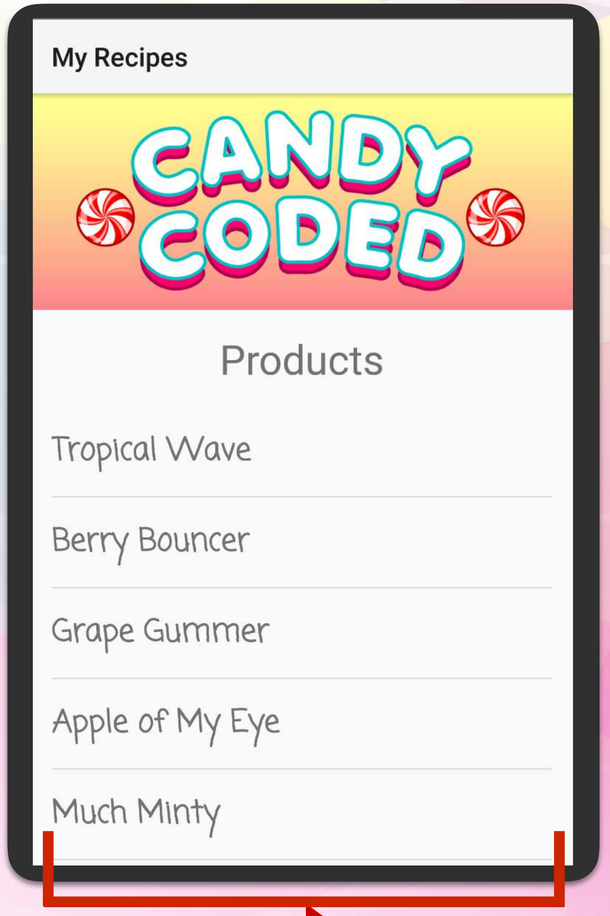
So it will be the height from the bottom of the `ImageView` *to the bottom of the screen*

If our list content is longer than the screen, it will automatically scroll

TRY ANDROID

What Does match_parent Mean?

The ListView**'s** layout_width **and** layout_height **are** match_parent**.**

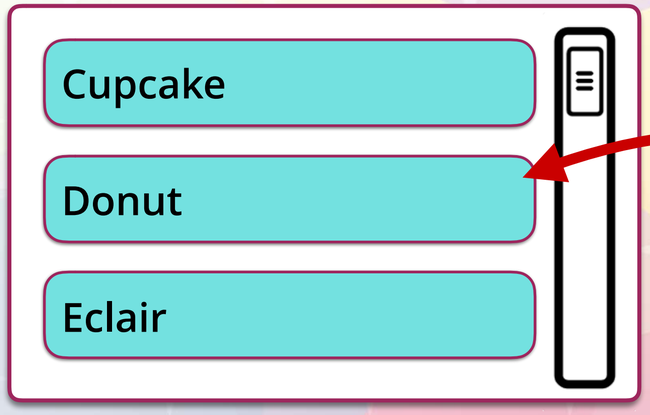


`match_parent` *as the* `layout_width` *means the view list will be the entire width of the screen*

TRY ANDROID

2 Creating a Layout for Each Item in Our ListView

We need a layout to tell the ArrayAdapter **how to build each item for the** ListView**.**



We'll use a single `TextView` *for each item in our* `ListView`

list_item_candy.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/text_view_candy">
</TextView>
```

`match_parent` *since we want each list item to fill the* `ListView`

We need an id to reference this later

Screencast: The Layout for the Items in Our ListView

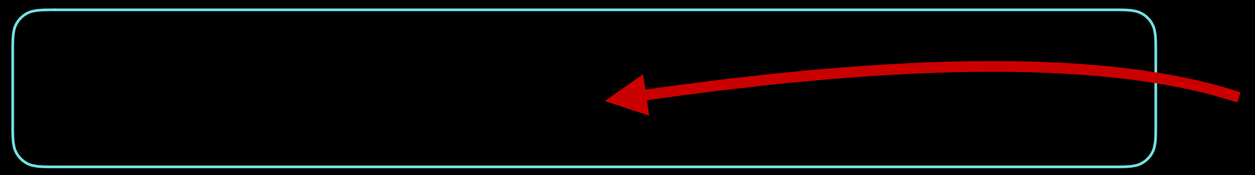
TRY ANDROID

Where to Add Code in MainActivity.java

We'll add our code to the bottom of the onCreate() **method, after we set the** TextView**'s text.**

MainActivity.java **Java**

```
...
public class MainActivity extends AppCompatActivity {

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

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

    }
}
```



We'll add all the code to create our `ListView` *here*

3 Creating the ArrayList of Raw Data

We need to create the list of raw data to display. We'll use an ArrayList for that.

MainActivity.java Java

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();

        candyList.add("Tropical Wave");
        candyList.add("Berry Bouncer");
        candyList.add("Grape Gummer");
        candyList.add("Apple of My Eye");
        ...
        candyList.add("ROYGBIV Spinner");
...
```

Creating an empty `ArrayList` *of* `Strings`

Adding `Strings` *to the list*

4 Creating the ArrayAdapter From an ArrayList

We need a few things in the ArrayAdapter **constructor: the** Activity**, the** Layout **for each item, the** TextView **for each item, and the** ArrayList **we just created.**

MainActivity.java **Java**

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                this,
                R.layout.list_item_candy,
                R.id.text_view_candy,
                candyList  );

...
```

We need these things in our ArrayAdapter *constructor to create our* ArrayAdapter

4 Creating the ArrayAdapter From an ArrayList

We need a few things in the ArrayAdapter **constructor: the** Activity**, the** Layout **for each item, the** TextView **for each item, and the** ArrayList **we just created.**

MainActivity.java **Java**

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                this,
                R.layout.list_item_candy,
                R.id.text_view_candy,
                candyList  );

...
```

Where this `ArrayAdapter` *will be used, or the context, is* `this` `Activity`

4 Creating the ArrayAdapter From an ArrayList

We need a few things in the ArrayAdapter **constructor: the** Activity**, the** Layout **for each item, the** TextView **for each item, and the** ArrayList **we just created.**

MainActivity.java **Java**

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                this,
                R.layout.list_item_candy,
                R.id.text_view_candy,
                candyList  );

...
```

The `Layout` *file we created to build each item in the* `ListView`

4 Creating the ArrayAdapter From an ArrayList

We need a few things in the ArrayAdapter **constructor: the** Activity**, the** Layout **for each item, the** TextView **for each item, and the** ArrayList **we just created.**

MainActivity.java **Java**

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                this,
                R.layout.list_item_candy,
                R.id.text_view_candy,
                candyList  );

...
```

We need the specific `TextView` *inside the* `Layout` *file. Remember we created an id for this previously...*

4 Creating the ArrayAdapter From an ArrayList

We need a few things in the ArrayAdapter **constructor: the** Activity**, the** Layout **for each item, the** TextView **for each item, and the** ArrayList **we just created.**

MainActivity.java **Java**

```java
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                this,
                R.layout.list_item_candy,
                R.id.text_view_candy,
                candyList );

...
```

Finally, the `ArrayList` *of our candies*

5 Connecting the ArrayAdapter to the ListView

To connect the ArrayAdapter **to the** ListView**, we first need to find the** ListView **so we have a reference to it. We can find** Views **with the** findViewById() **method.**

MainActivity.java **Java**

```
import java.util.ArrayList;

...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                ... );

        ListView listView = (ListView)this.findViewById(
                R.id._____________);

...
```

Remember we created an id for the ListView *in the* MainActivity*'s Layout*

The id for the ListView in main_activity.xml

We need the id **for the** ListView **so we can find it and then attach our** ArrayAdapter **to it.**

activity_main.xml XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <TextView …/> <ImageView …/>
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list_view_candy"/>
</LinearLayout>
```

MainActivity.java

```
...
    ListView listView = (ListView)this.findViewById(
            R.id.list_view_candy);
...
```

5 Connecting the ArrayAdapter to the ListView

Finally! We can connect the ListView **to the** ArrayAdapter **and see our** ListView **in action.**

MainActivity.java **Java**

```
...
        ArrayList<String> candyList = new ArrayList<String>();
        ...

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                ... );

        ListView listView = (ListView)this.findViewById(
                R.id.list_view_candy);

        listView.setAdapter(adapter);
...
```

Now that we found the ListView *by its* id, *we can set its* Adapter

Screencast: Demo All the Code in MainActivity.java


TRY
ANDROID

Level 4

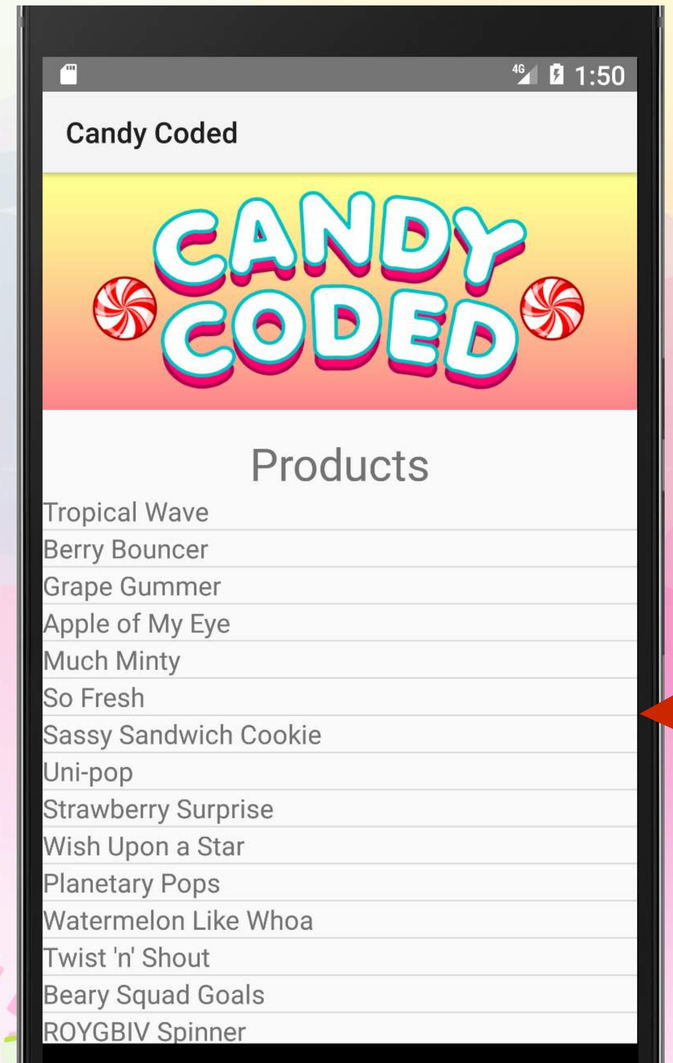
EventListeners & Toasts

Making Our ListView Do Something

TRY ANDROID

Improving the ListView's Look

We want to make the ListView look better by adding padding and updating the font and text size for the list items.

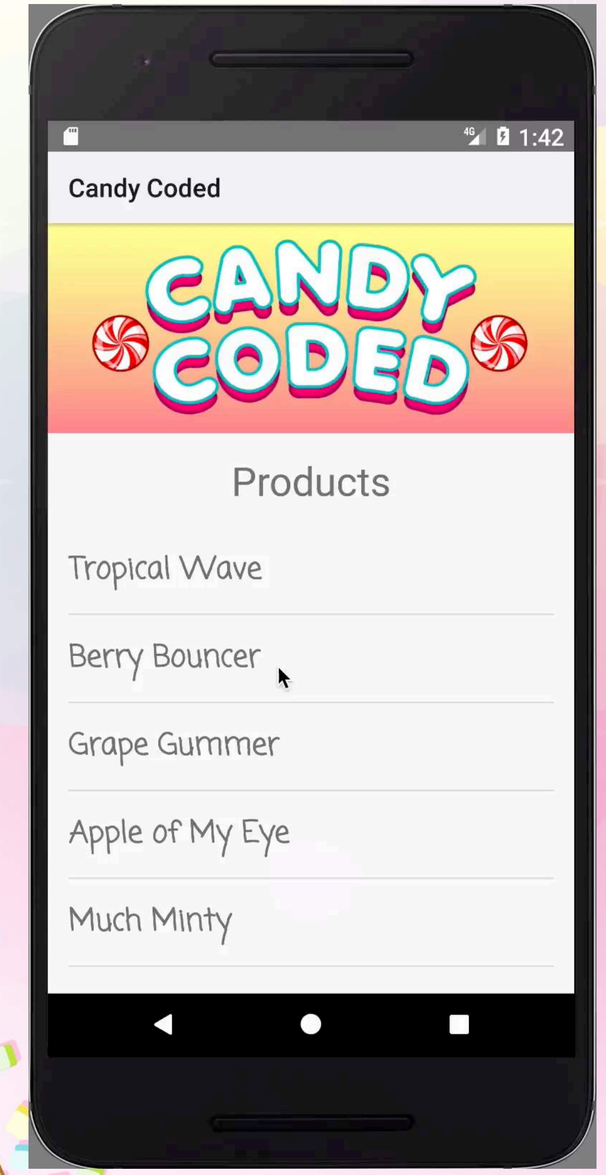


The `ListView` and its items need padding

TRY ANDROID

Screencast: Adding Padding to the ListView

Making Our List Items Do Something



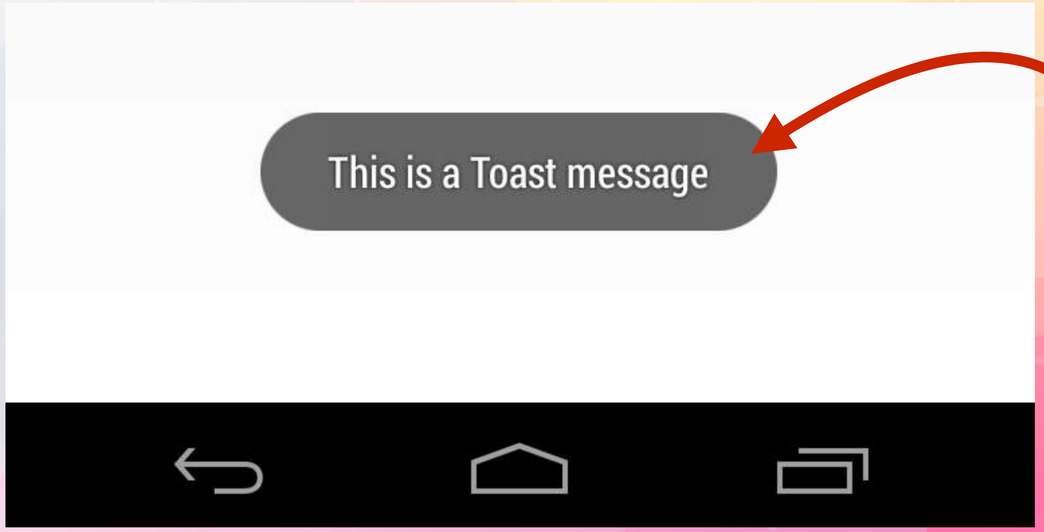
Eventually, we'd like to open a detail screen when the user clicks on a candy

As a first step, we'll display the position of the item in the list when the user clicks on it

TRY ANDROID

Let's Display a Toast

A Toast is a small popup message, and the current activity remains visible and interactive.



We can still see and interact with the `Activity` *below the* `Toast`

TRY ANDROID

Creating Our First Toast

First, let's create a simple Toast when our app loads, and then we'll add one for clicks on the list items.

MainActivity.java

```
...
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        listView.setAdapter(adapter);

        // After connecting the ListView and Adapter


    }
}
```

How to Make a Toast

To create a Toast**, we need to know the** Context**,** text**, and** duration**.**

MainActivity.java

```
...
    Context context = this;
    String text = "Hello toast!";
    int duration = Toast.LENGTH_SHORT;

    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
...
```

Creates the `Toast`

Displays the `Toast`

The `context` *is where this* `Toast` *will run, which for us is the* `MainActivity`*.*

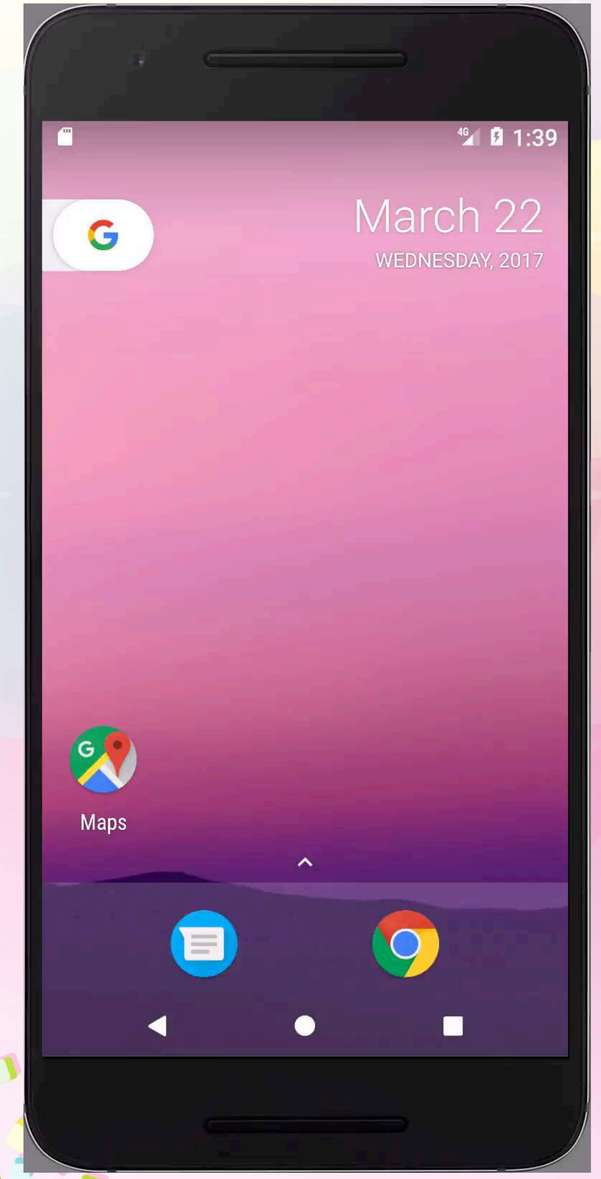
So we can just use `this` *since we're in the* `MainActivity` *class.*

The `text` *is whatever message you want to display*

The `duration` *is either* `Toast.LENGTH_SHORT` *or* `Toast.LENGTH_LONG`

TRY ANDROID

Demo of Our First Toast



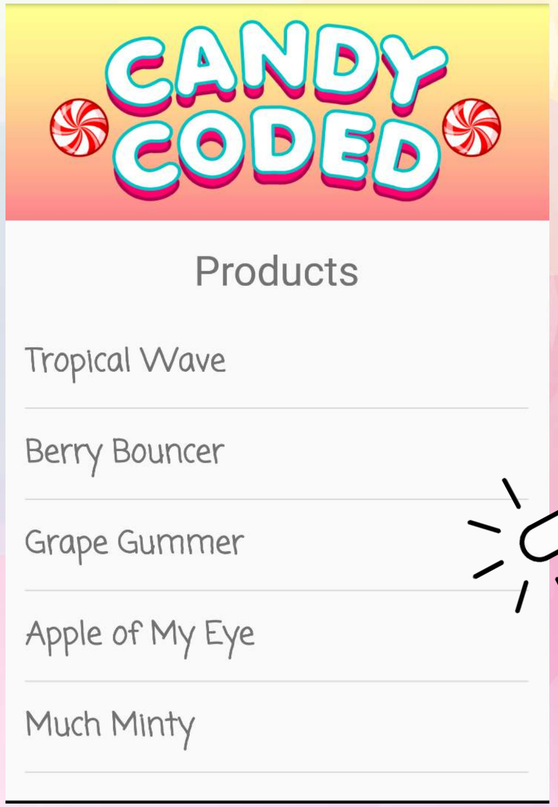
We can see our "Hello Toast!" Toast comes up when the app starts.

Now we want to show a `Toast` *when a list item is clicked!*

TRY ANDROID

EventListeners

To trigger a Toast **on a click event, we can use an** EventListener**. An** EventListener **contains a single callback method, which gets called when the user triggers the event.**



User clicks item

Listener receives click event

Listener calls the callback method

```
public void onItemClick(
    // Do something
)
```

TRY ANDROID

Adding an EventListener

An EventListener **contains a single callback method, which gets called when the user triggers the event. In this case,** onItemClick() **will get called when the user touches an item in the list.**

Our `ListView` *Set its* `ItemClickListener`

```
listView.setOnItemClickListener(




);
```

TRY ANDROID

Adding an EventListener

An EventListener **contains a single callback method, which gets called when the user triggers the event. In this case,** onItemClick() **will get called when the user touches an item in the list.**

Setting the item's click listener a new ItemClickListener

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {




});
```

TRY ANDROID

Adding an EventListener

An EventListener **contains a single callback method, which gets called when the user triggers the event. In this case,** onItemClick() **will get called when the user touches an item in the list.**

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view,
                            int i, long l) {

        // Do something here

    }
});
```

This is the callback method that gets called when the user touches an item in our list

TRY ANDROID

Adding an EventListener

We'll add our EventListener **code to the bottom of our** onCreate() **method again.**

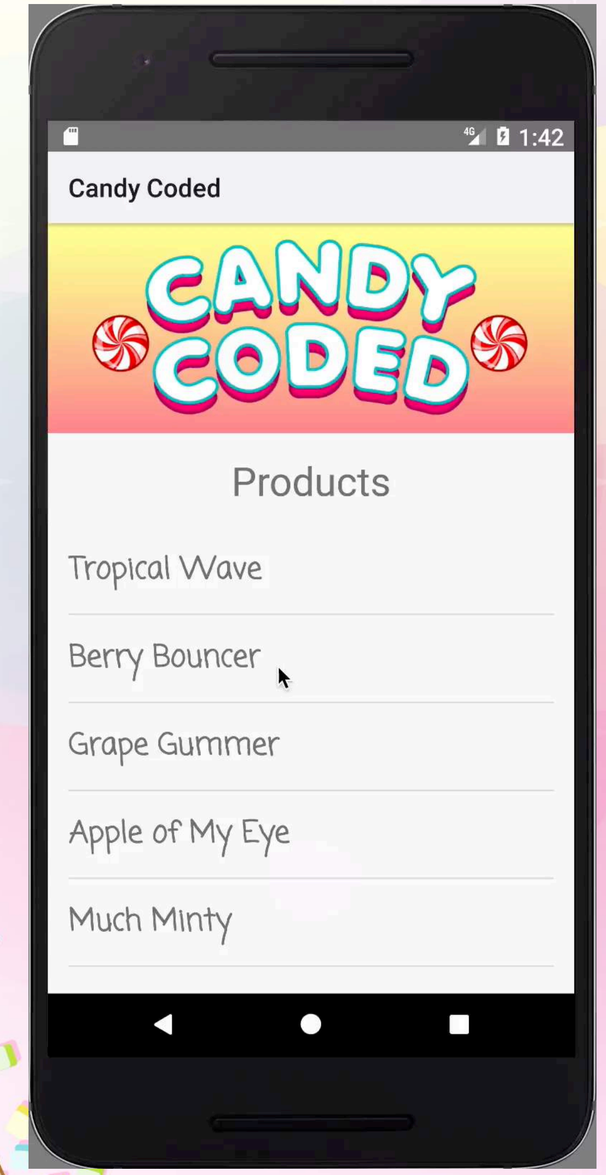
MainActivity.java

```
...
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view,
                            int i, long l) {

        Toast toast = Toast.makeText(MainActivity.this, ""+i, Toast.LENGTH_SHORT);
        toast.show();
    }
});
...
```

Let's add a `Toast` *to show the item's position* `i`

We need to specify `MainActivity.this` *as the context, instead of just* `this`*, since we're inside the* `ClickListener`

Demoing the ClickListener & Toast in Action



Now we can see our list items actually doing something!

TRY ANDROID


TRY
ANDROID