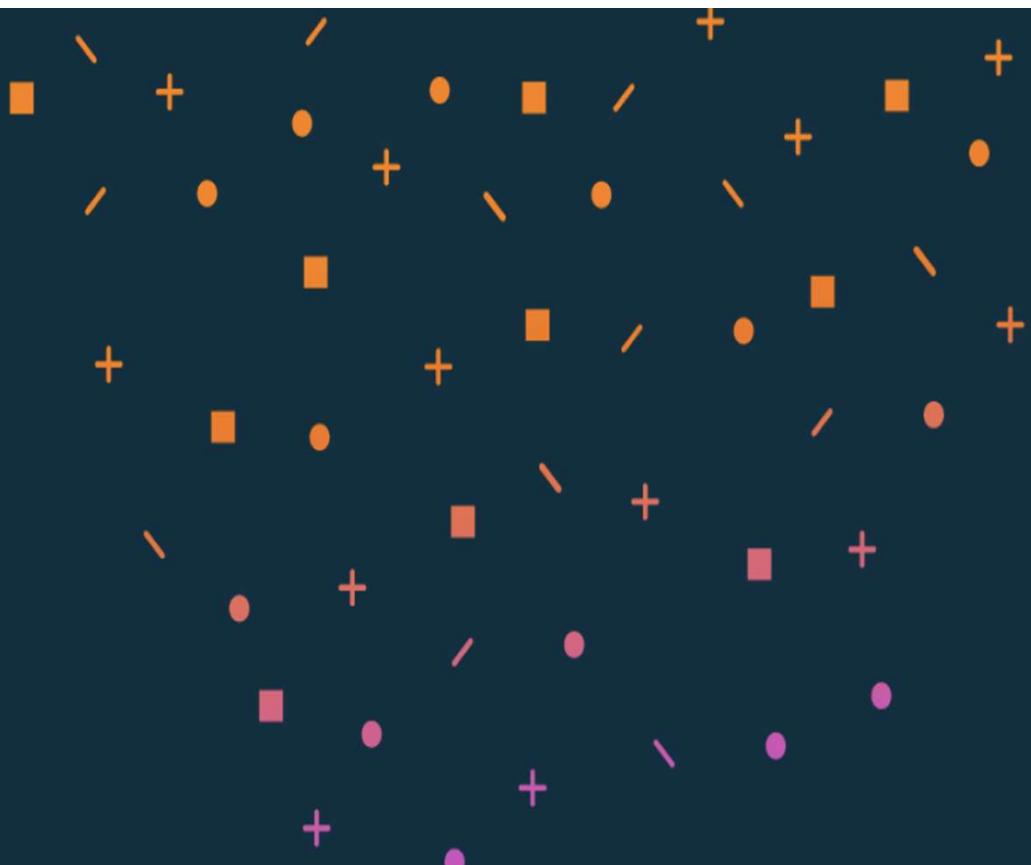


Lesson 5:

Layouts



1

About this lesson

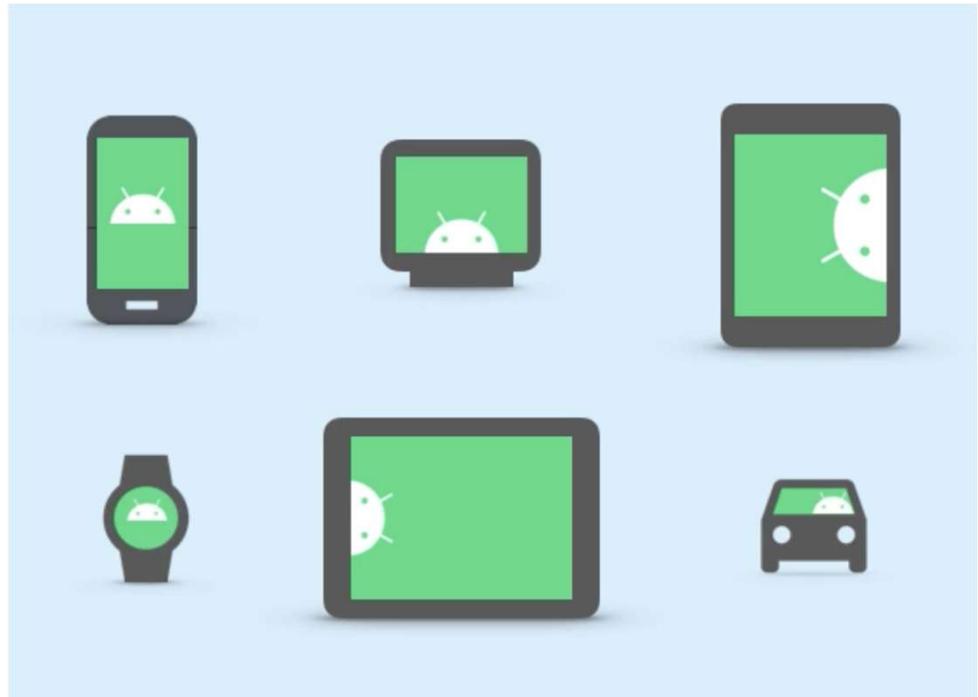
Lesson 5: Layouts

- [Layouts in Android](#)
- [ConstraintLayout](#)
- [Additional topics for ConstraintLayout](#)
- [Data binding](#)
- [Displaying lists with RecyclerView](#)
- [Summary](#)

Layouts in Android

Android devices

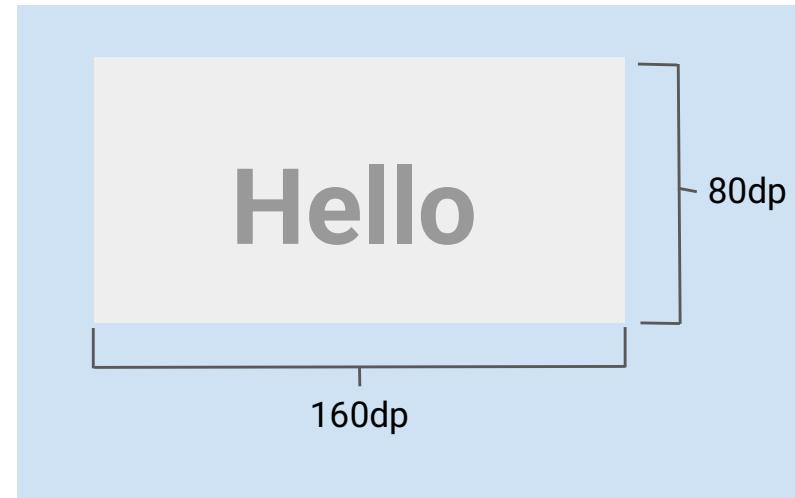
- Android devices come in many different form factors.
- More and more pixels per inch are being packed into device screens.
- Developers need the ability to specify layout dimensions that are consistent across devices.



Density-independent pixels (dp)

Use dp when specifying sizes in your layout, such as the width or height of views.

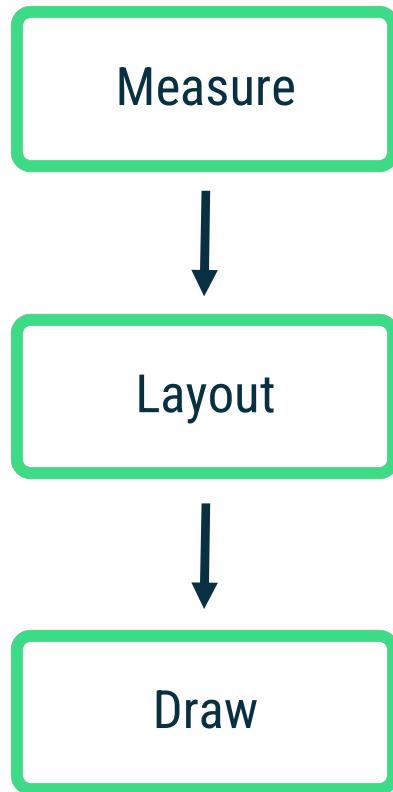
- Density-independent pixels (dp)
take screen density into account.
- Android views are measured in
density-independent pixels.
- $dp = \frac{\text{(width in pixels} * 160)}{\text{screen density}}$



Screen-density buckets

Density qualifier	Description	DPI estimate
ldpi (mostly unused)	Low density	~120dpi
mdpi (baseline density)	Medium density	~160dpi
hdpi	High density	~240dpi
xhdpi	Extra-high density	~320dpi
xxhdpi	Extra-extra-high density	~480dpi
xxxhdpi	Extra-extra-extra-high density	~640dpi

Android View rendering cycle



Drawing region

What we see:

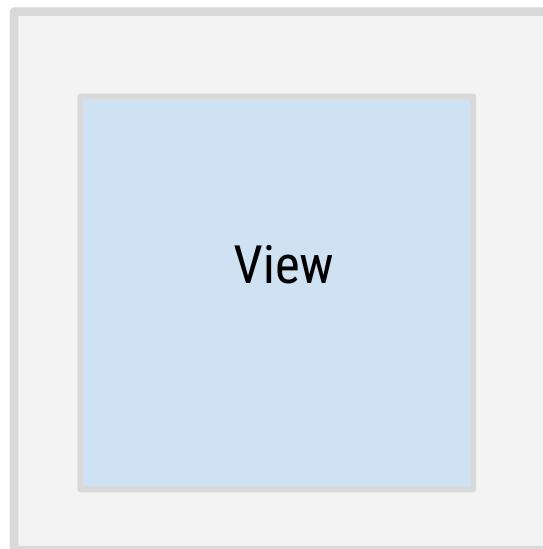


How it's drawn:

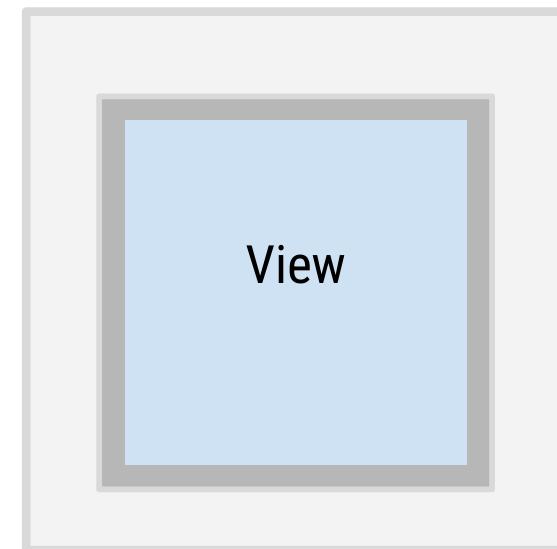


View margins and padding

View with margin



View with margin and padding



ConstraintLayout

Deeply nested layouts are costly

- Deeply nested ViewGroups require more computation
- Views may be measured multiple times
- Can cause UI slowdown and lack of responsiveness

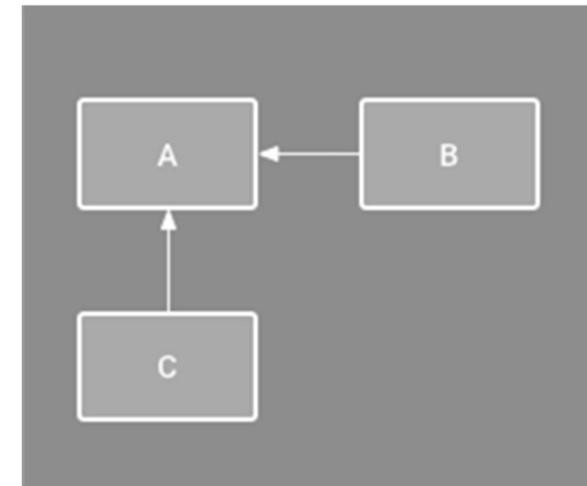
Use ConstraintLayout to avoid some of these issues!

What is ConstraintLayout?

- Recommended default layout for Android
- Solves costly issue of too many nested layouts, while allowing complex behavior
- Position and size views within it using a set of constraints

What is a constraint?

A restriction or limitation on
the properties of a View that
the layout attempts to
respect



Relative positioning constraints

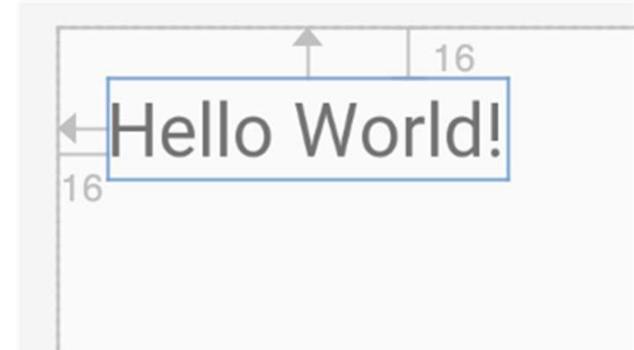
Can set up a constraint relative to the parent container

Format: `layout_constraint<SourceConstraint>_to<TargetConstraint>Of`

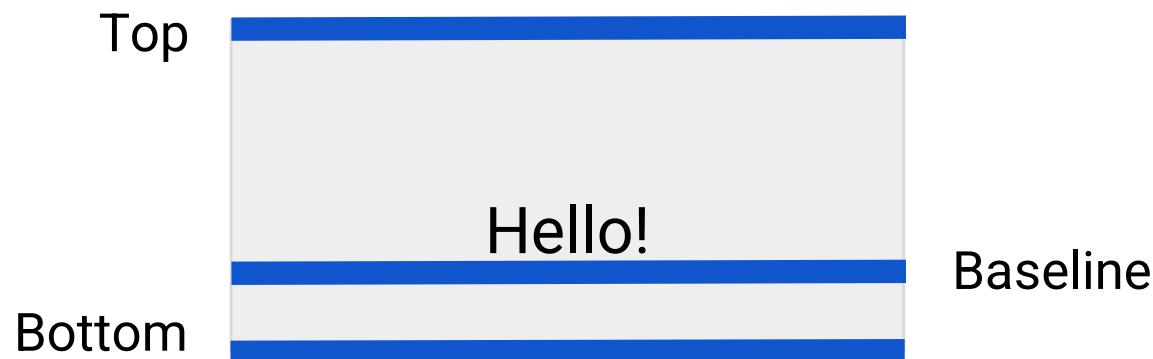
Example attributes on a `TextView`:

`app:layout_constraintTop_toTopOf="parent"`

`app:layout_constraintLeft_toLeftOf="parent"`



Relative positioning constraints

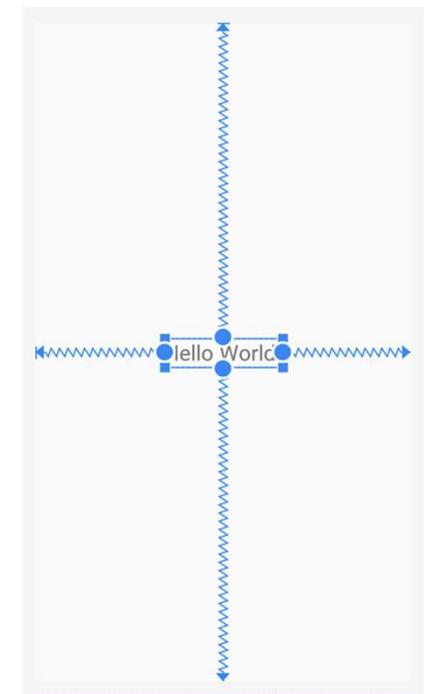


Relative positioning constraints



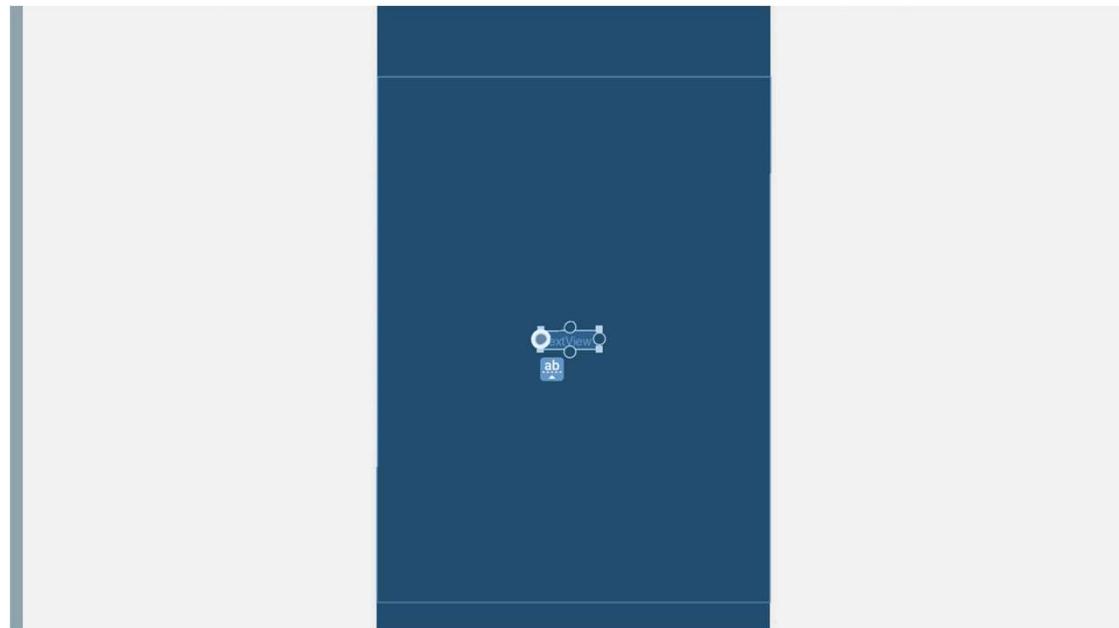
Simple ConstraintLayout example

```
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
<TextView  
    ...  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```



Layout Editor in Android Studio

You can click and drag to add constraints to a View.



Constraint Widget in Layout Editor



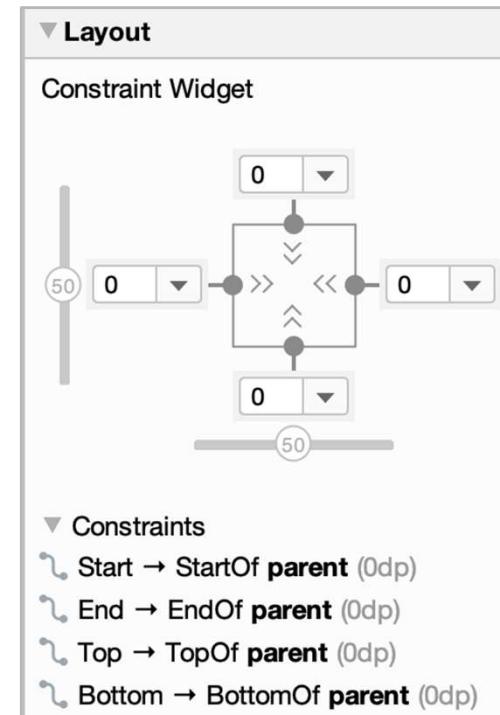
Fixed



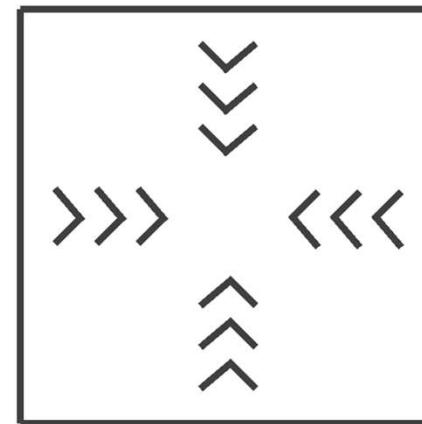
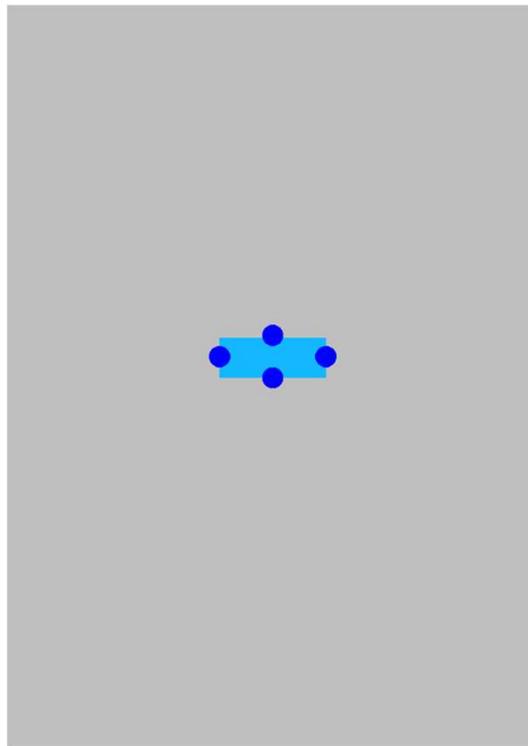
Wrap content



Match constraints



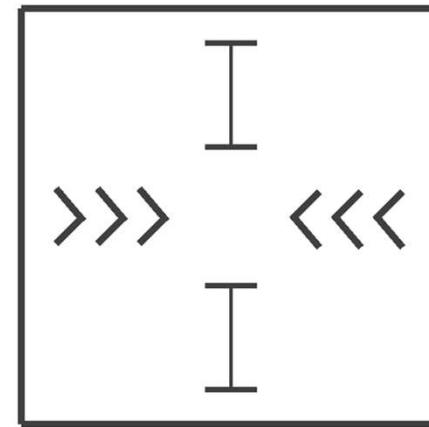
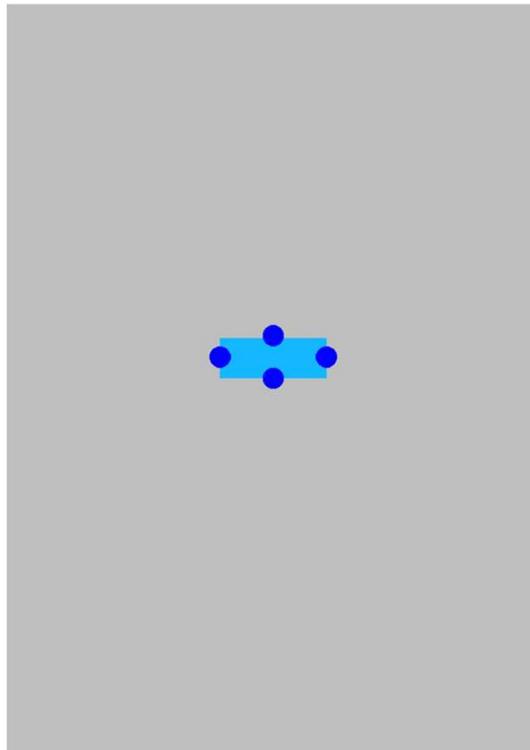
Wrap content for width and height



layout_width wrap_content

layout_height wrap_content

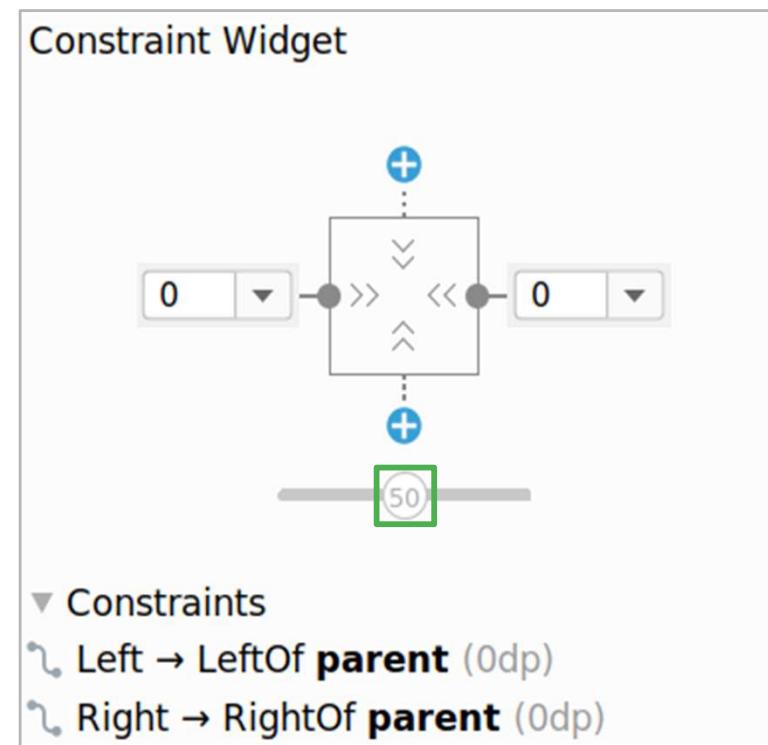
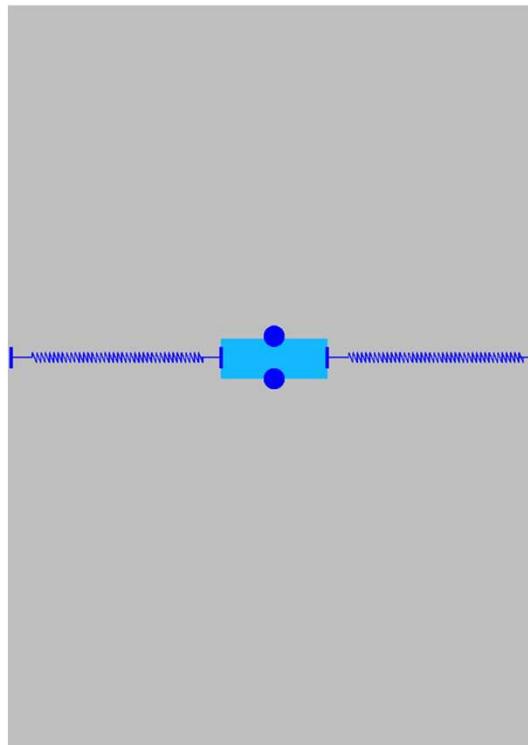
Wrap content for width, fixed height



layout_width wrap_content

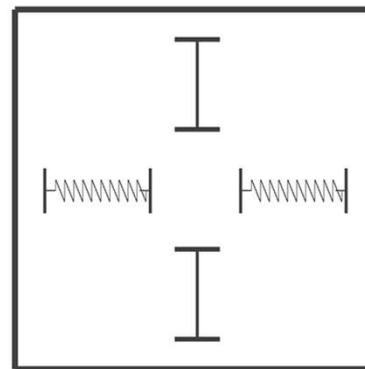
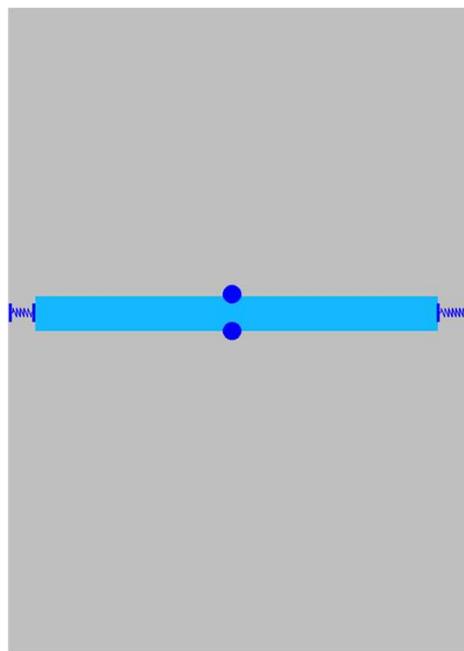
layout_height 48dp

Center a view horizontally



Use match_constraint

Can't use match_parent on a child view, use match_constraint instead



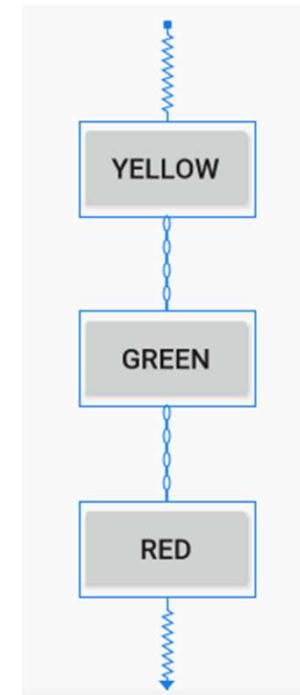
```
layout_width    0dp(match_constraint)  
layout_height   48dp
```

Chains

- Let you position views in relation to each other
- Can be linked horizontally or vertically
- Provide much of LinearLayout functionality

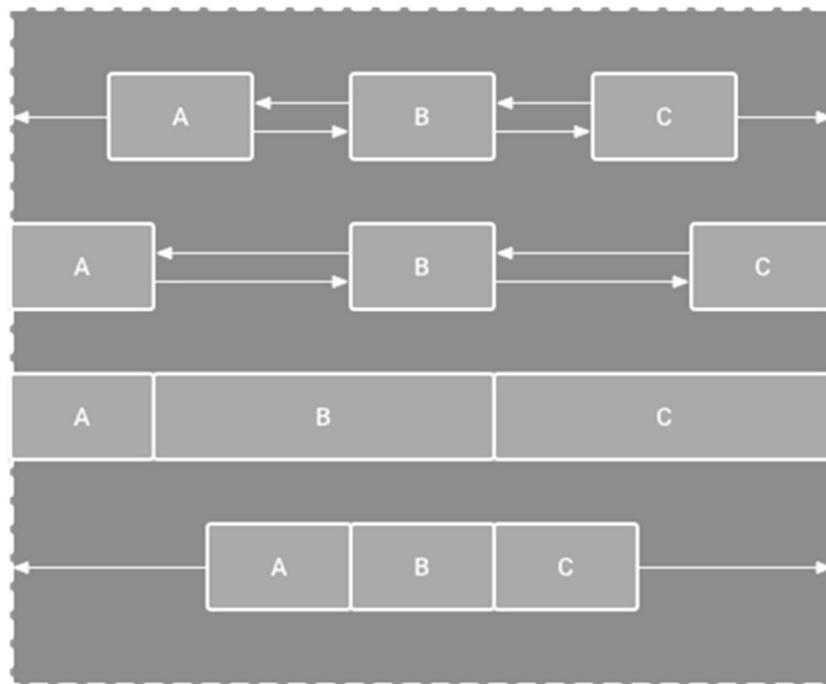
Create a Chain in Layout Editor

- Select the objects you want to be in the chain.
- Right-click and select **Chains**.
- Create a horizontal or vertical chain.



Chain styles

Adjust space between views with these different chain styles.



Spread Chain

Spread Inside Chain

Weighted Chain

Packed Chain

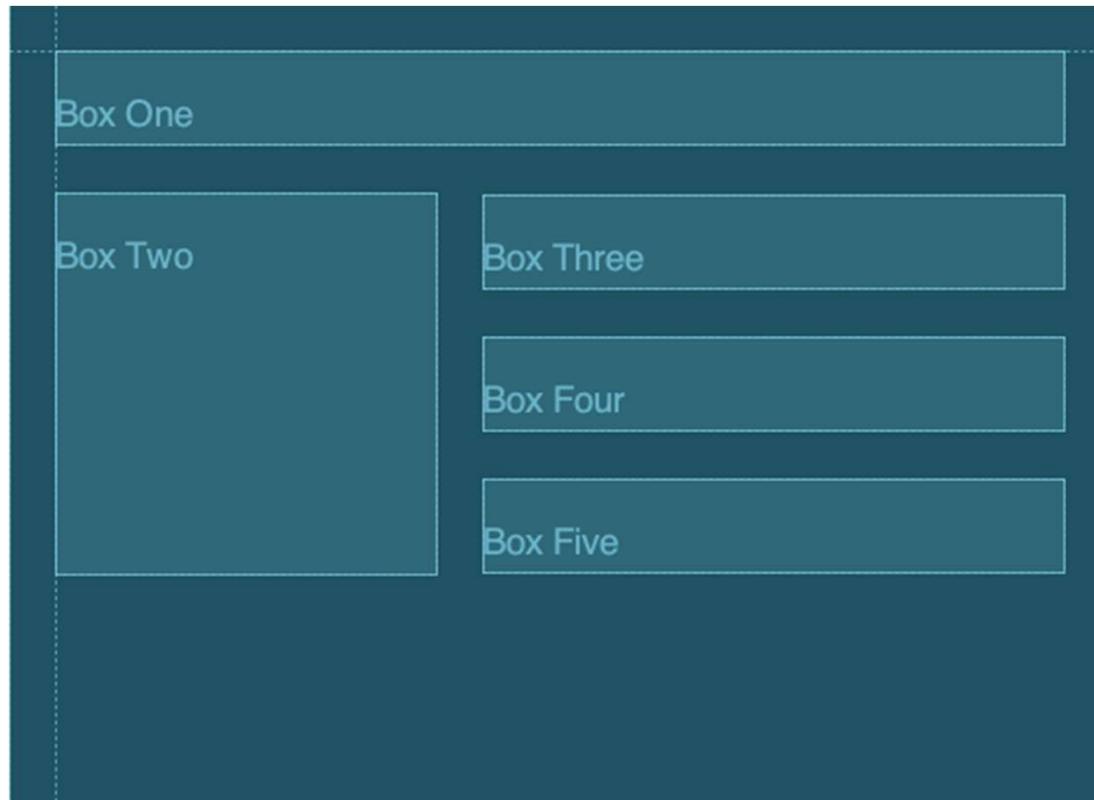
Additional topics for ConstraintLayout

27

Guidelines

- Let you position multiple views relative to a single guide
- Can be vertical or horizontal
- Allow for greater collaboration with design/UX teams
- Aren't drawn on the device

Guidelines in Android Studio



Example Guideline

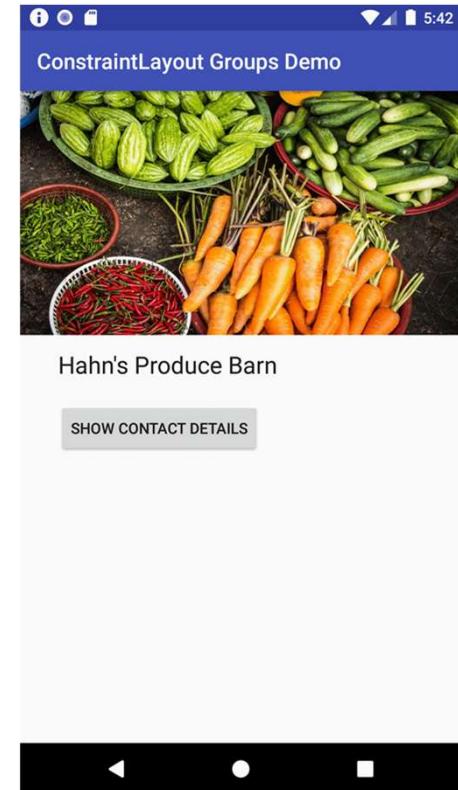
```
<ConstraintLayout>
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/start_guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_begin="16dp" />
    <TextView ...
        app:layout_constraintStart_toEndOf="@id/start_guideline" />
</ConstraintLayout>
```

Creating Guidelines

- `layout_constraintGuide_begin`
- `layout_constraintGuide_end`
- `layout_constraintGuide_percent`

Groups

- Control the visibility of a set of widgets
- Group visibility can be toggled in code



Example group

```
<androidx.constraintlayout.widget.Group  
    android:id="@+id/group"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:constraint_referenced_ids="locationLabel,locationDetails"/>
```

Groups app code

```
override fun onClick(v: View?) {  
    if (group.visibility == View.GONE) {  
        group.visibility = View.VISIBLE  
        button.setText(R.string.hide_details)  
    } else {  
        group.visibility = View.GONE  
        button.setText(R.string.show_details)  
    }  
}
```

Data binding

Current approach: findViewById()

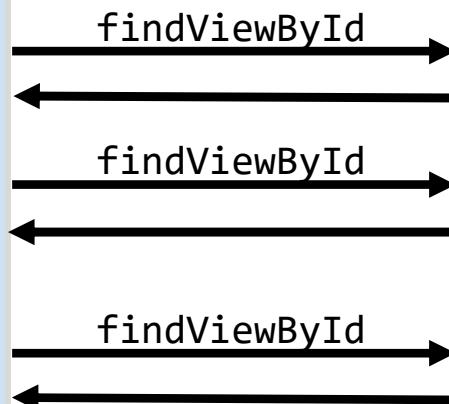
Traverses the View hierarchy each time

MainActivity.kt

```
val name = findViewById<TextView>(...)  
val age = findViewById<TextView>(...)  
val loc = findViewById<TextView>(...)  
  
name.text = ...  
age.text = ...  
loc.text = ...
```

activity_main.xml

```
<ConstraintLayout ... >  
  <TextView  
    android:id="@+id/name"/>  
  <TextView  
    android:id="@+id/age"/>  
  <TextView  
    android:id="@+id/loc"/>  
</ConstraintLayout>
```

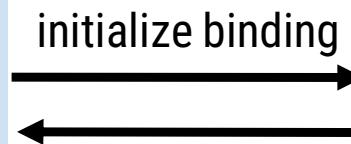


Use data binding instead

Bind UI components in your layouts to data sources in your app.

MainActivity.kt

```
Val binding:ActivityMainBinding  
  
binding.name.text = ...  
binding.age.text = ...  
binding.loc.text = ...
```



activity_main.xml

```
<layout>  
  <ConstraintLayout ... >  
    <TextView  
      android:id="@+id/name"/>  
    <TextView  
      android:id="@+id/age"/>  
    <TextView  
      android:id="@+id/loc"/>  
  </ConstraintLayout>  
</layout>
```

Modify build.gradle file

```
android {  
    ...  
    buildFeatures {  
        dataBinding true  
    }  
}
```

Add layout tag

```
<layout>

    <androidx.constraintlayout.widget.ConstraintLayout>
        <TextView ... android:id="@+id/username" />
        <EditText ... android:id="@+id/password" />
    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

Layout inflation with data binding

Replace this

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

with this

```
val binding: ActivityMainBinding = DataBindingUtil.setContentView(  
    this, R.layout.activity_main)
```

```
binding.username = "Melissa"
```

Data binding layout variables

```
<layout>
    <data>
        <variable name="name" type="String"/>
    </data>
    <androidx.constraintlayout.widget.ConstraintLayout>
        <TextView
            android:id="@+id/textView"
            android:text="@{name}" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

In MainActivity.kt:

```
binding.name = "John"
```

Data binding layout expressions

```
<layout>
    <data>
        <variable name="name" type="String"/>
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout>
        <TextView
            android:id="@+id/textView"
            android:text="@{name.toUpperCase()}" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Displaying lists with RecyclerView

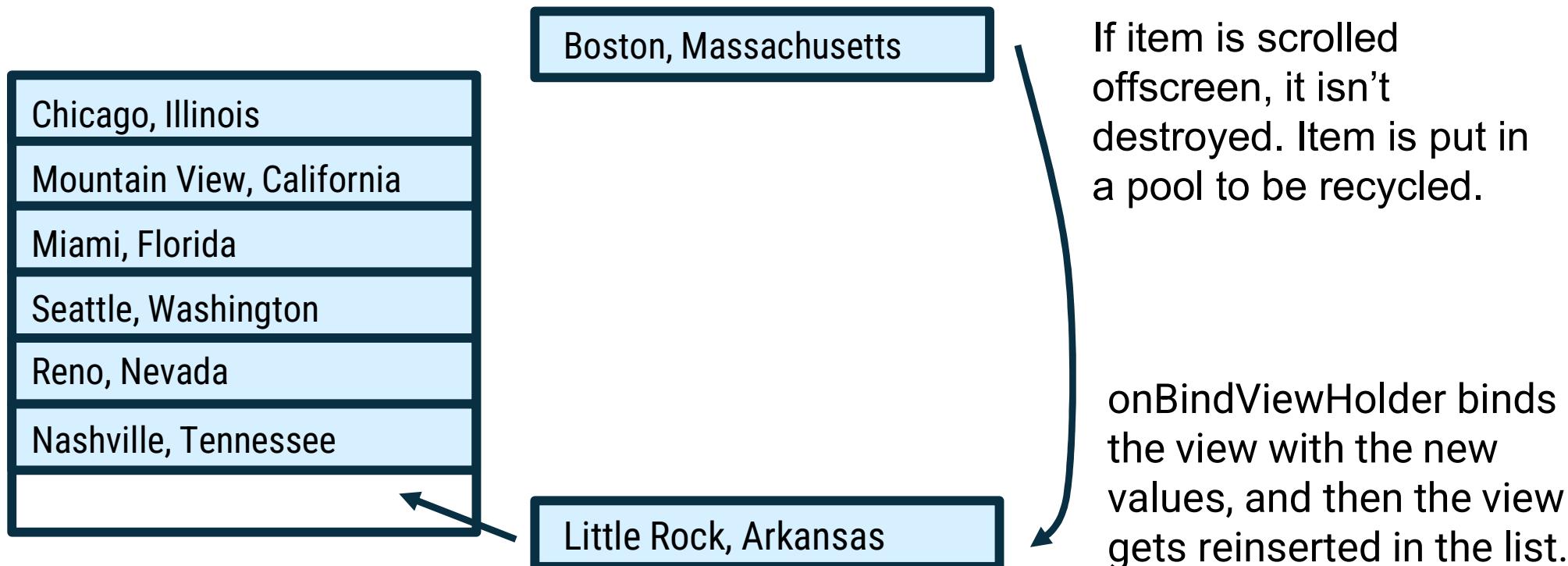
RecyclerView

- Widget for displaying lists of data
- "Recycles" (reuses) item views to make scrolling more performant
- Can specify a list item layout for each item in the dataset
- Supports **animations and transitions**

RecyclerView.Adapter

- Supplies data and layouts that the RecyclerView displays
- A custom Adapter extends from `RecyclerView.Adapter` and overrides these three functions:
 - `getItemCount`
 - `onCreateViewHolder`
 - `onBindViewHolder`

View recycling in RecyclerView



Add RecyclerView to your layout

```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/rv"  
    android:scrollbars="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

Create a list item layout

res/layout/item_view.xml

```
<FrameLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
    <TextView  
        android:id="@+id/number"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content" />  
</FrameLayout>
```

Create a list adapter

```
class MyAdapter(val data: List<Int>) : RecyclerView.Adapter<MyAdapter.MyViewHolder>() {
    class MyViewHolder(val row: View) : RecyclerView.ViewHolder(row) {
        val textView = row.findViewById<TextView>(R.id.number)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val layout = LayoutInflater.from(parent.context).inflate(R.layout.item_view,
            parent, false)
        return MyViewHolder(layout)
    }

    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
        holder.textView.text = data.get(position).toString()
    }

    override fun getItemCount(): Int = data.size
}
```

Set the adapter on the RecyclerView

In MainActivity.kt:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    val rv: RecyclerView = findViewById(R.id.rv)  
    rv.layoutManager = LinearLayoutManager(this)  
  
    rv.adapter = MyAdapter(IntRange(0, 100).toList())  
}
```

Summary

Summary

In Lesson 5, you learned how to:

- Specify lengths in dp for your layout
- Work with screen densities for different Android devices
- Render Views to the screen of your app
- Layout views within a ConstraintLayout using constraints
- Simplify getting View references from layout with data binding
- Display a list of text items using a RecyclerView and custom adapter

Learn more

- [Pixel density on Android](#)
- [Spacing](#)
- [Device metrics](#)
- [Type scale](#)
- [Build a Responsive UI with ConstraintLayout](#)
- [Data Binding Library](#)
- [Create dynamic lists with RecyclerView](#)

Pathway

Practice what you've learned by
completing the pathway:

[Lesson 5: Layouts](#)

