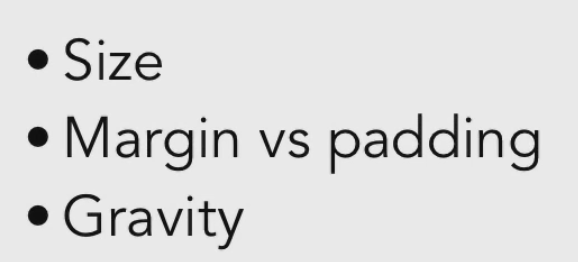
**Learning Outcomes**

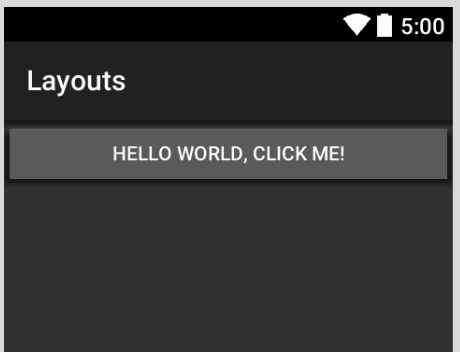
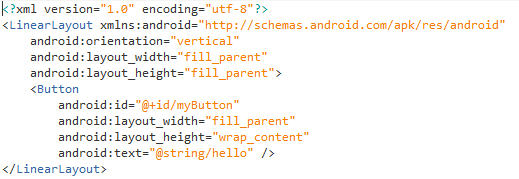
1. Layout fundamentals
2. Creating Responsive Layouts
3. Handling Device Orientation

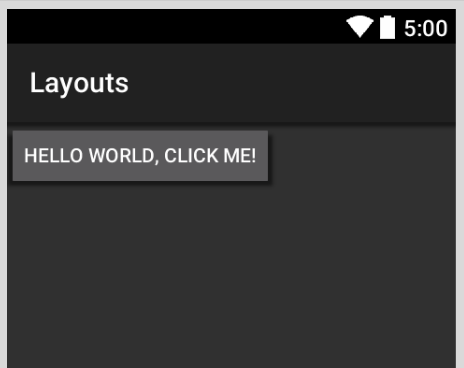
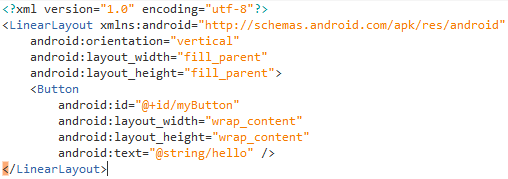
**Design Principles**

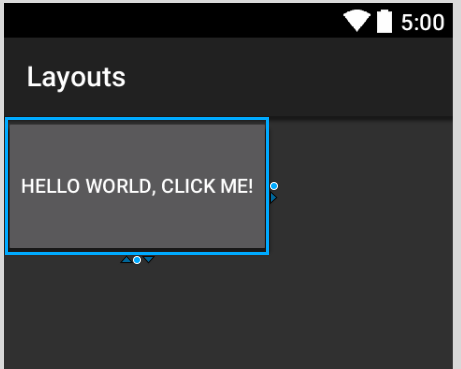
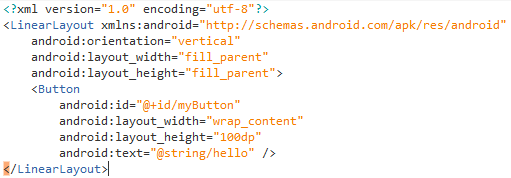
**Basic Attributes**



Fill\_Parent and Wrap\_Content

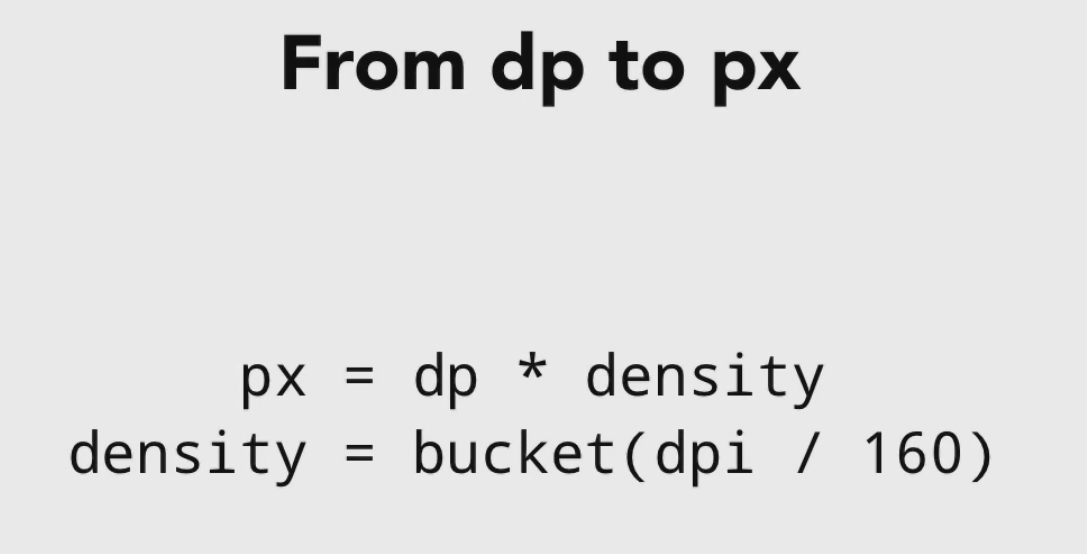






**What is “DP”**





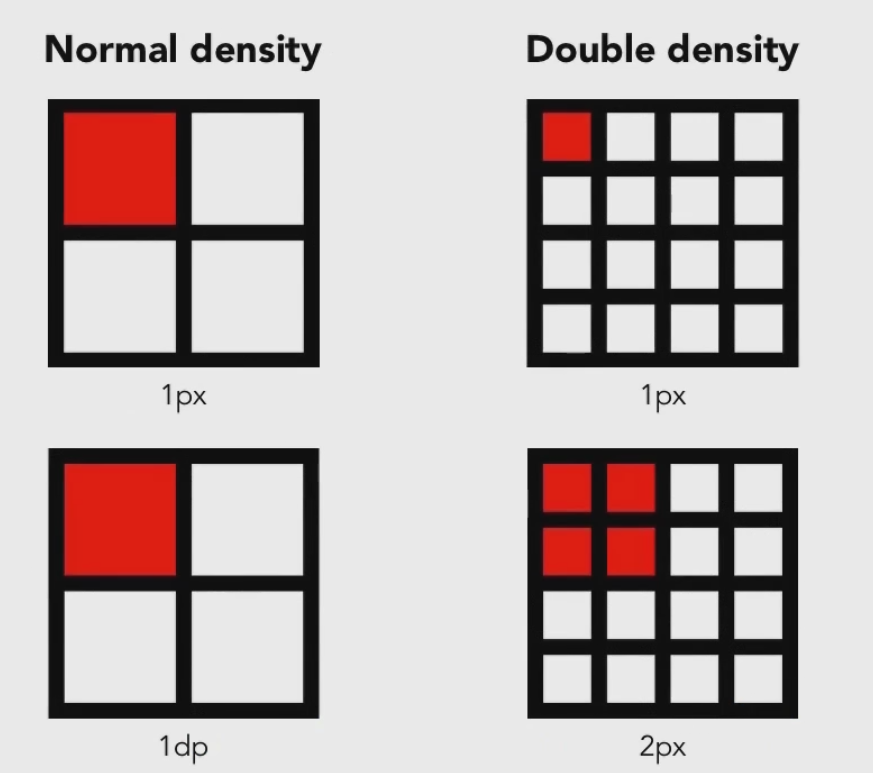
When Android was first launched it was on a device with 160 dpi, later different devices came into the market with different density and dpi values.

160 became the base value for density which is mdpi (medium density) with a multiplication factor of 1.

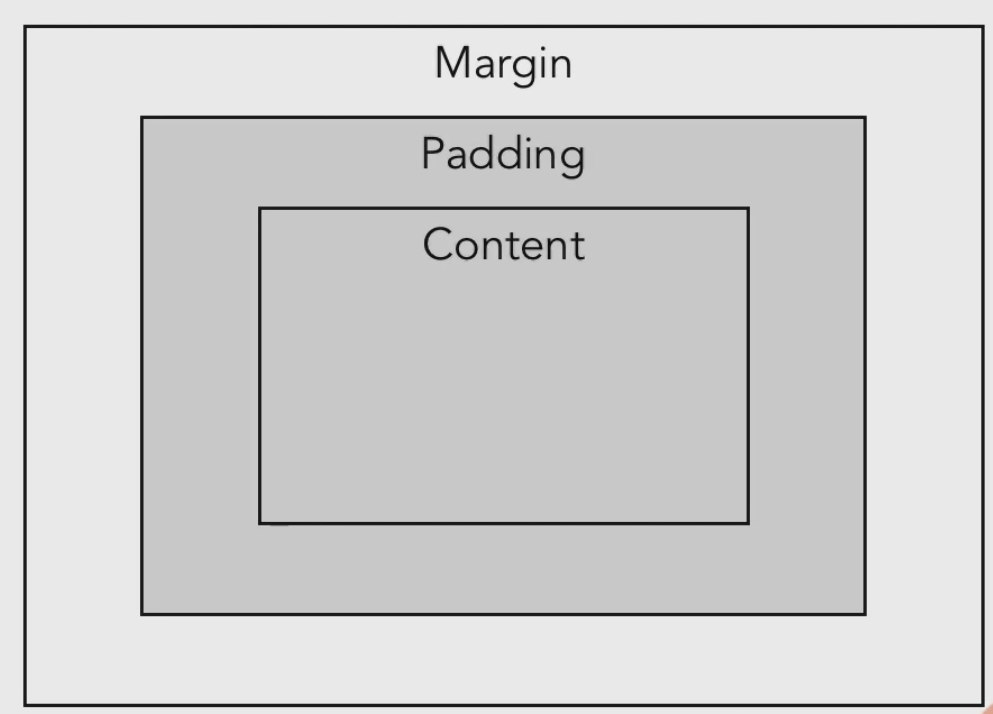
So 100 dp on mdpi device is 100 \* 1 = 100 px

100 dp on hdpi device is 100 \* 1.5 = 150 px



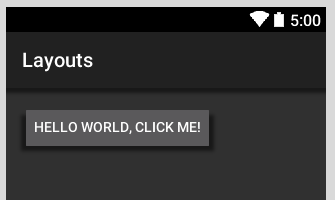


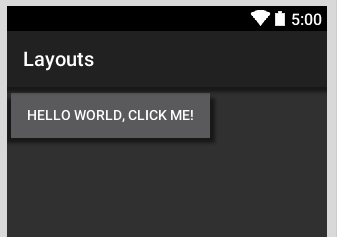
**Margin and Padding**



Padding - Space within the view

Margin - Space between the parent and the view

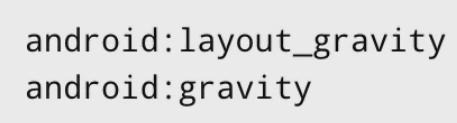
 





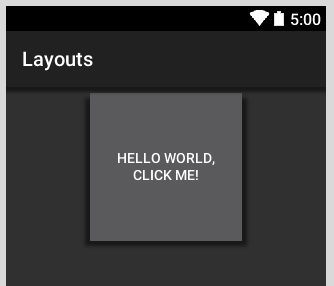
If the attribute has the keyword layout, it specifies it’s relation with the parent

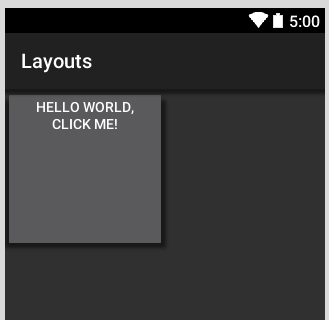
**Layout Gravity and Gravity**



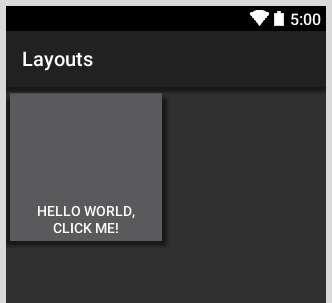
Layout\_Gravity – position of the view with respect to its parent

Gravity – position of the content within the view

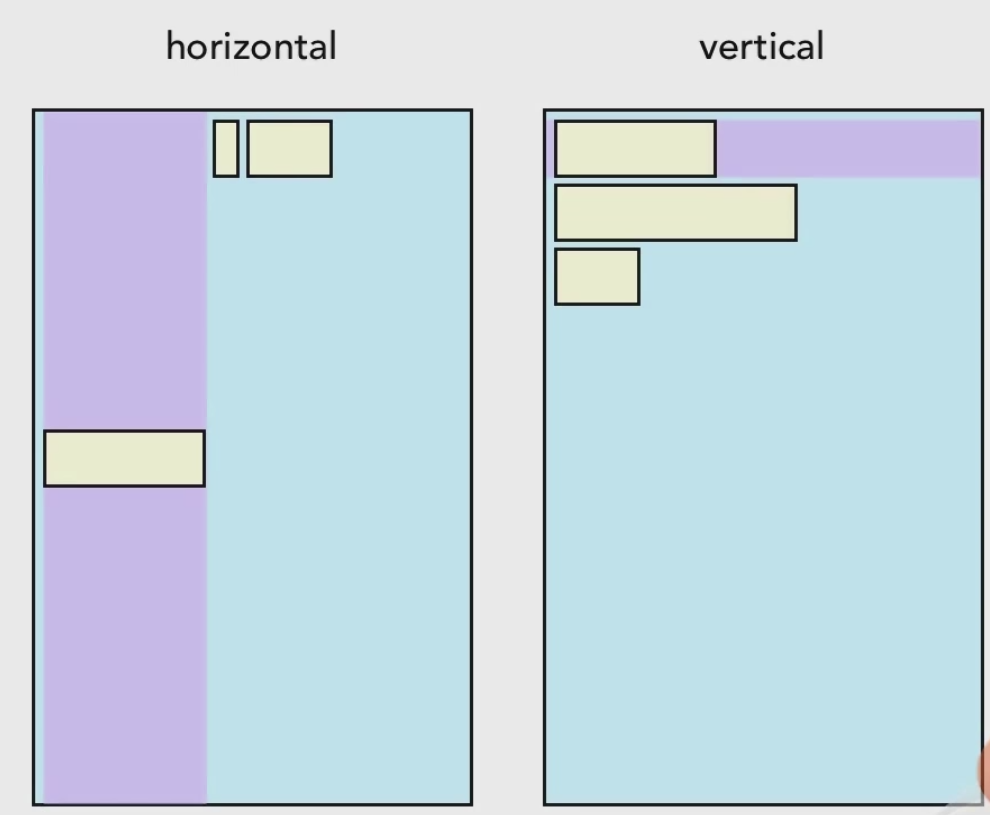




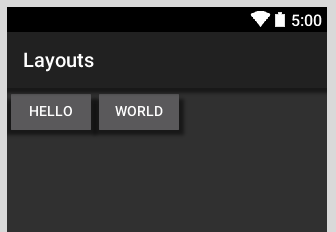
**Specifying more than one attribute**



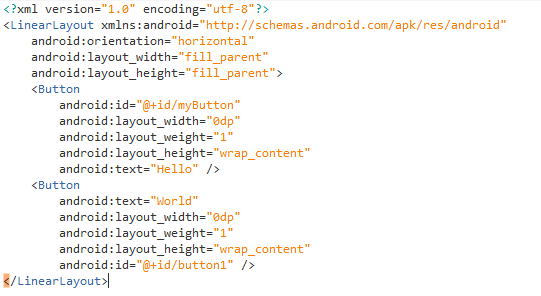
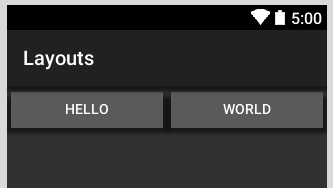
**Linear Layout**



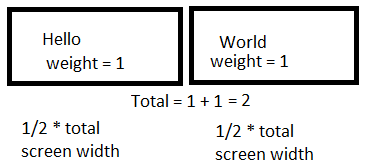
android:orientation = “horizontal” or “verticle”

Assigning weights to make the view proportionate and take space with respect to the layout.

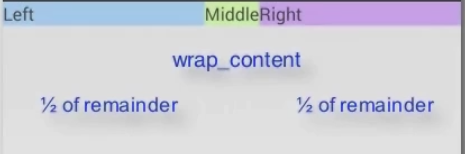
 

So how does android calculate the width from the weight?

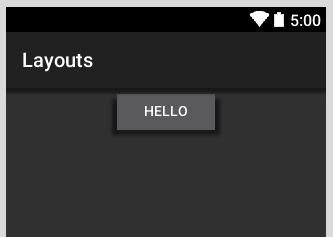
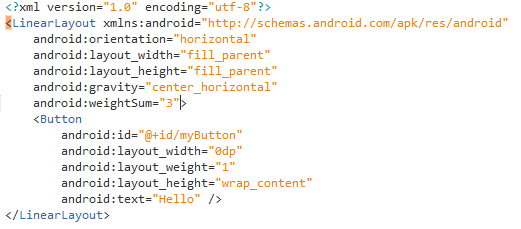


Similarly for three views with layout\_weight = “1”



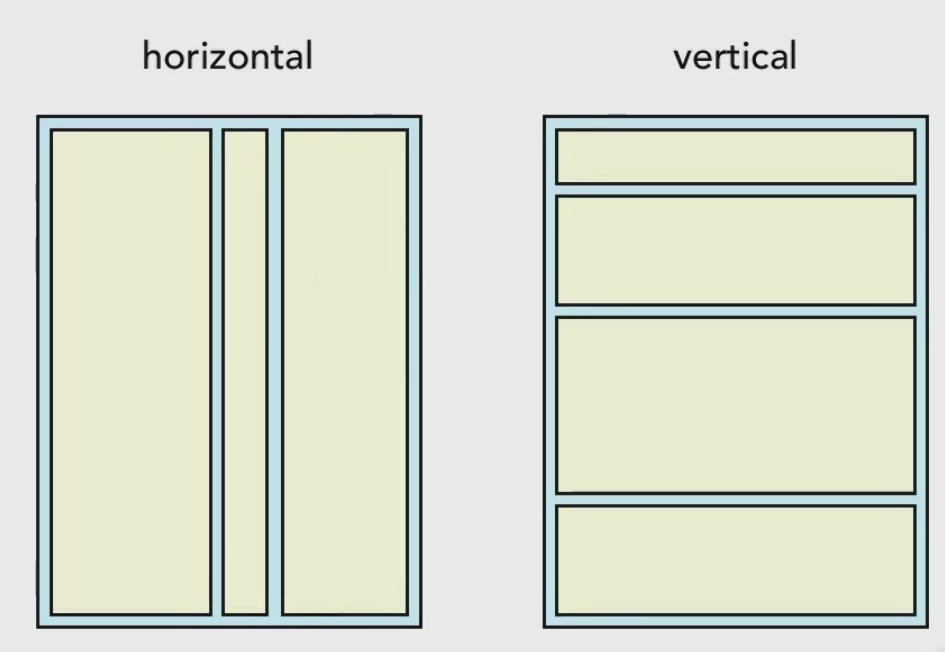


**weight\_sum**

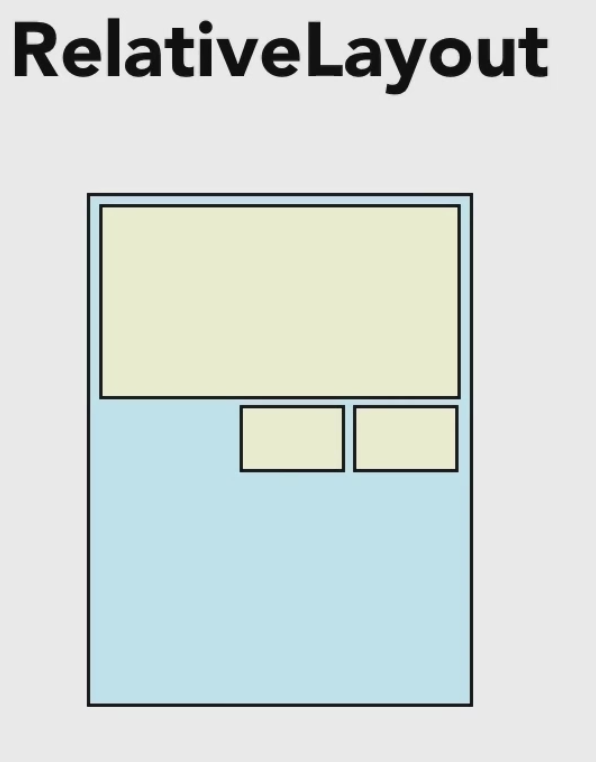


**Basic Layouts**

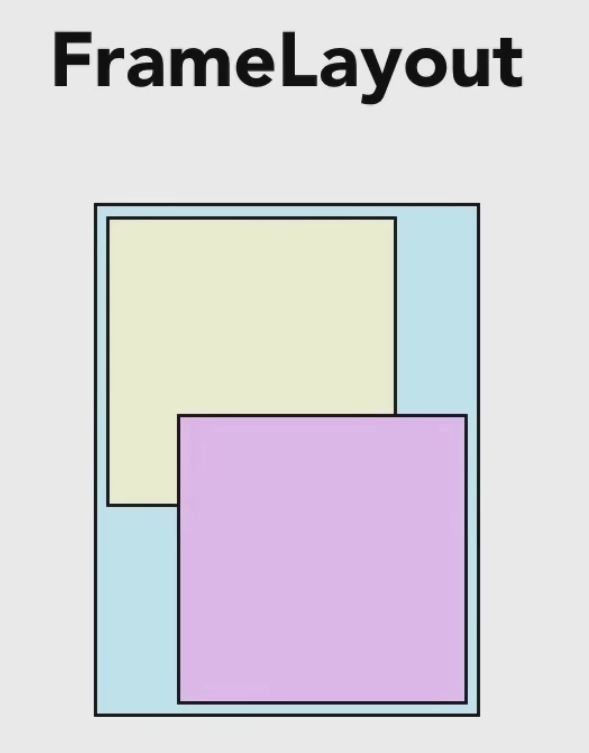
## **Linear Layout**



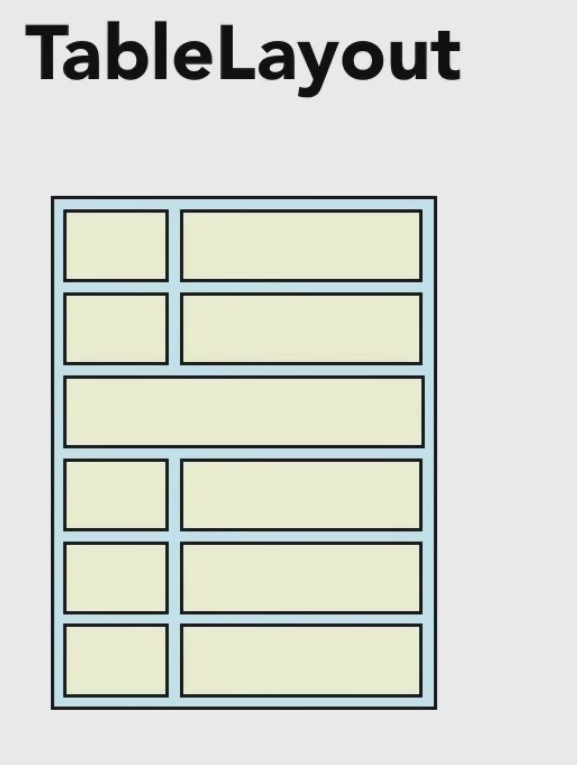
**Relative Layout**



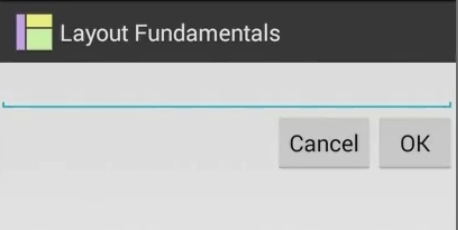
**Frame Layout**



**Table Layout**

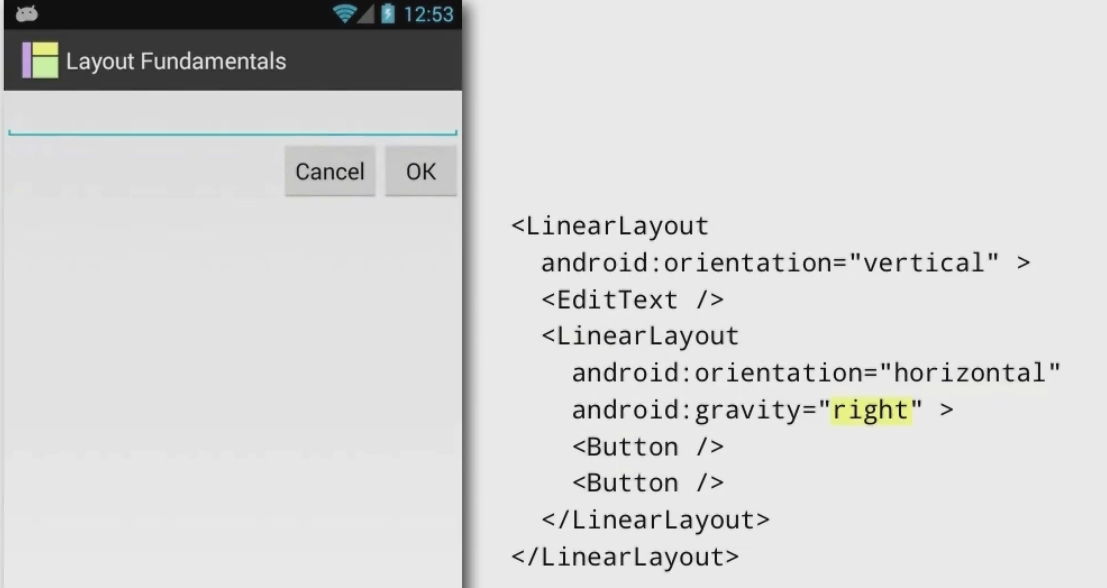


**Nested Linear layout**



Solution a nested linear layout

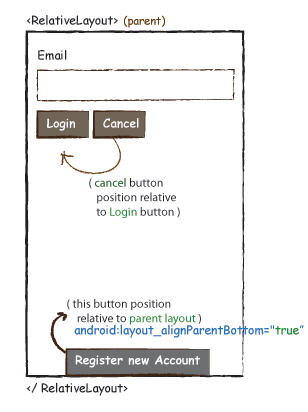




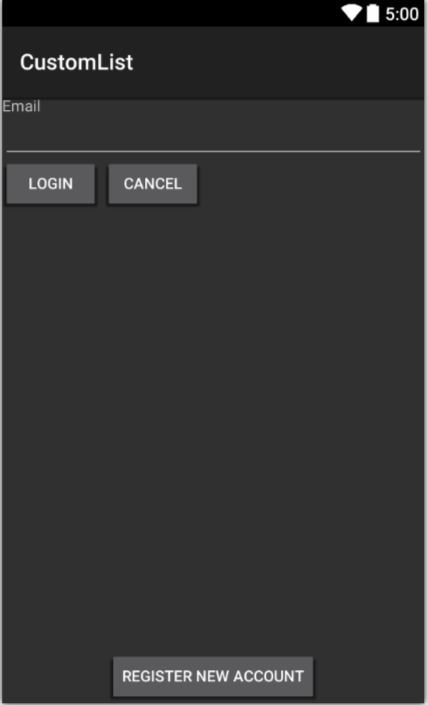
**Relative Layout**

In a relative layout every element arranges itself relative to other elements or a parent element.  
As an example, let’s consider the layout defined below. The “Cancel” button is placed relatively, to the right of the “Login” button parallely. Here is the code snippet that achieves the mentioned alignment (Right of Login button parallely).







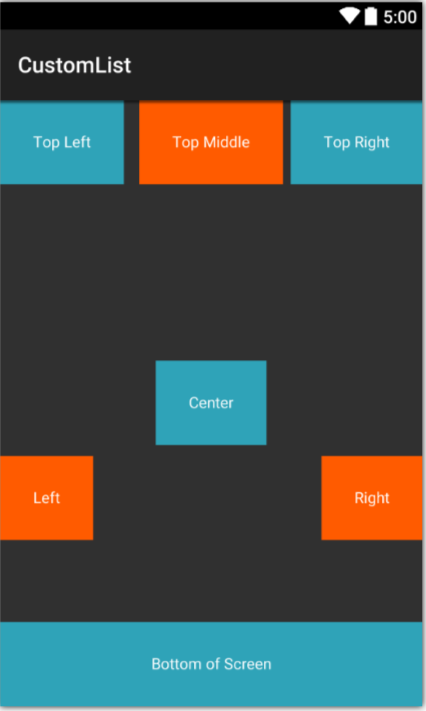


**Another example for Relative Layout**

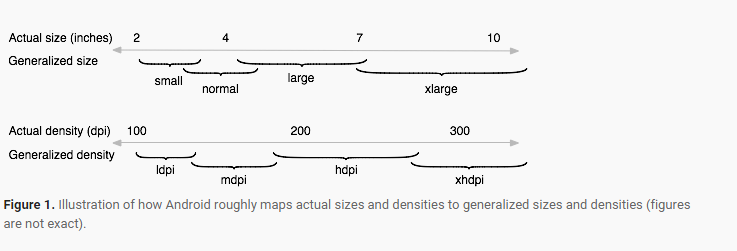








**Supporting multiple screen sizes**



### Using Alignment to Control Width or Height

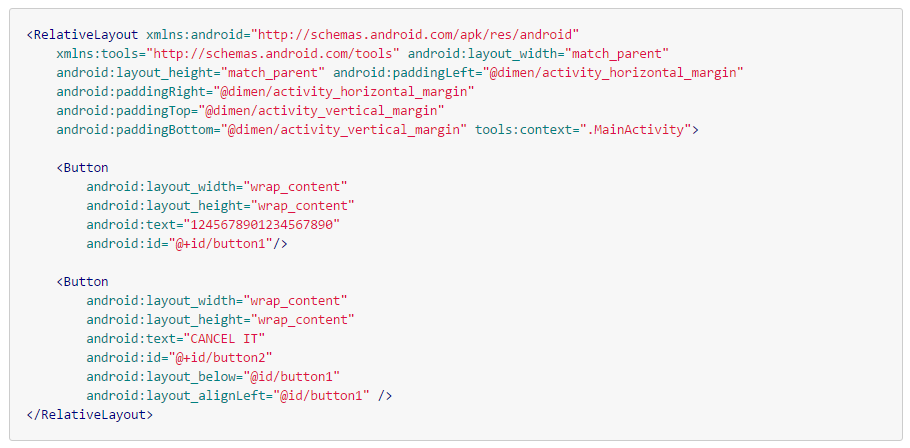
A less understood aspect of RelativeLayout is how the use of alignment can determine width or height. It may seem counterintuitive at first about how this works, so we'll walkthrough a few examples in this section. Using this approach is especially useful in matching the widths or heights relative to other elements.

#### Example 1: How alignment can determine width

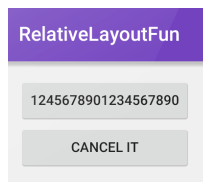
Suppose we have two buttons of varying widths:



The corresponding XML would be:



Suppose we also specify that the second button should be aligned left **and** right to the first button. If we add android:layout\_alignRight="@id/button1" to the second button's XML style, the change causes the second button to expand the width to match that of the first button. In other words, the only way to meet the requirements of specifying alignment on both sides is to expand the width of the second button.



In this way, when two elements are vertically positioned above or below the other, left and right alignments will control the **width**. When two elements are positioned horizontally next to each other, top and bottom alignments will control the **height**. We'll show how height can be impacted by specifying top and bottom alignments in the next example.

#### Example 2: How alignment can determine height

Suppose we have this layout definition:



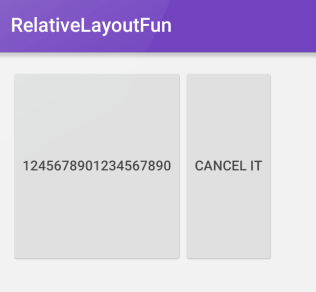
The corresponding preview looks like:

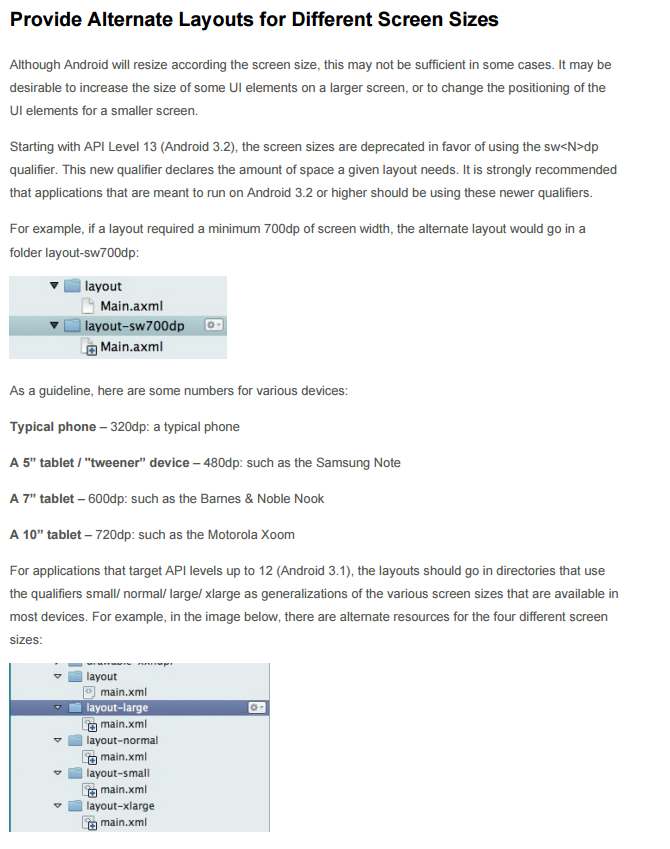


If we wish to match the height of the first button, we can specify layout\_alignTop and layout\_alignBottom on the second button.

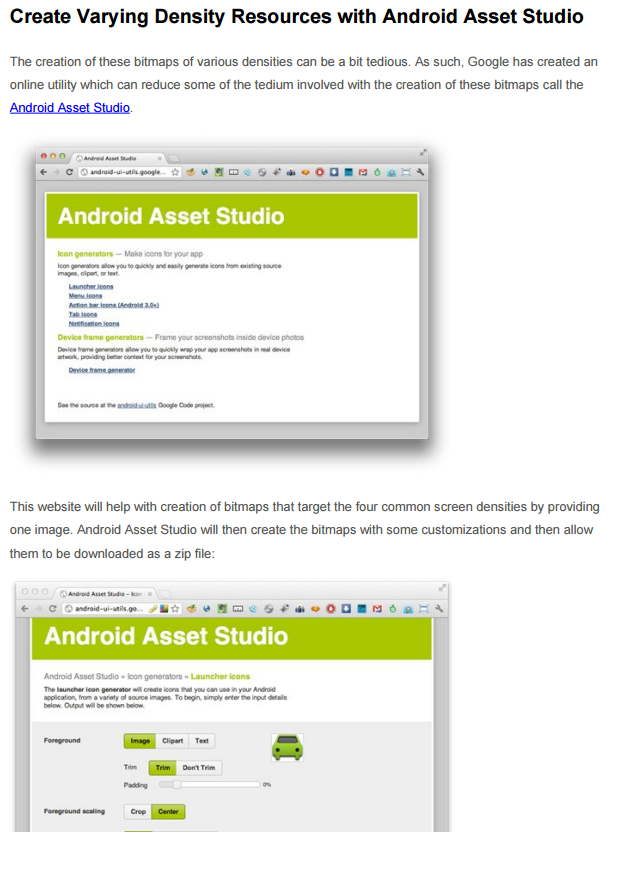


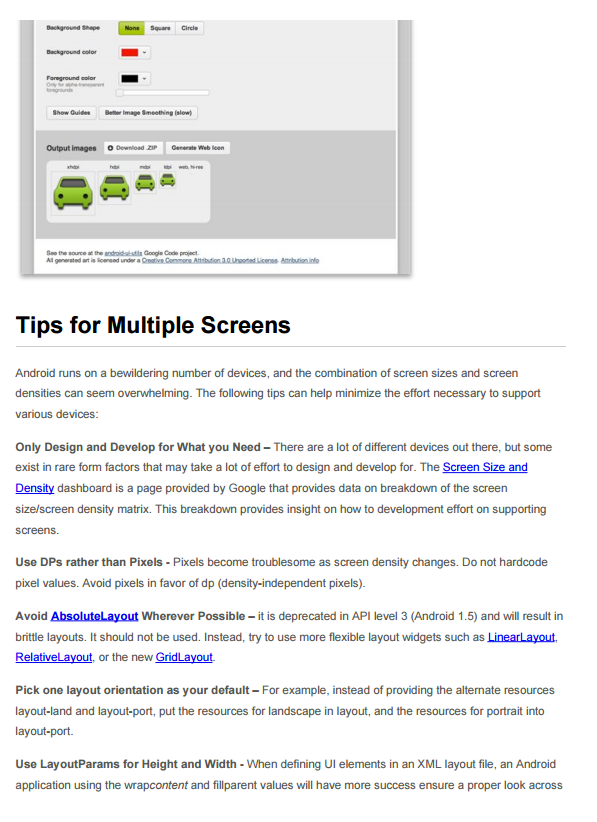
The only way to fulfill this requirement is to expand the height of the second button as shown below:

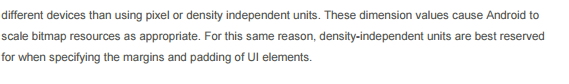




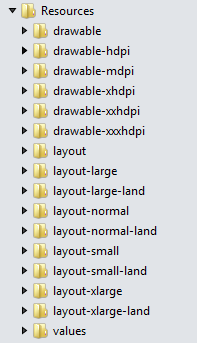








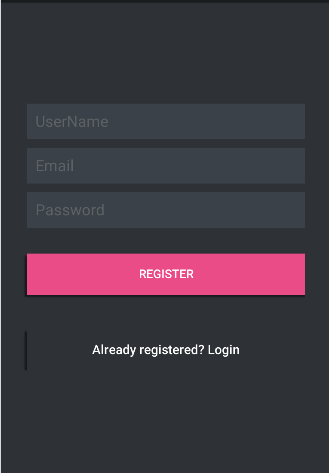
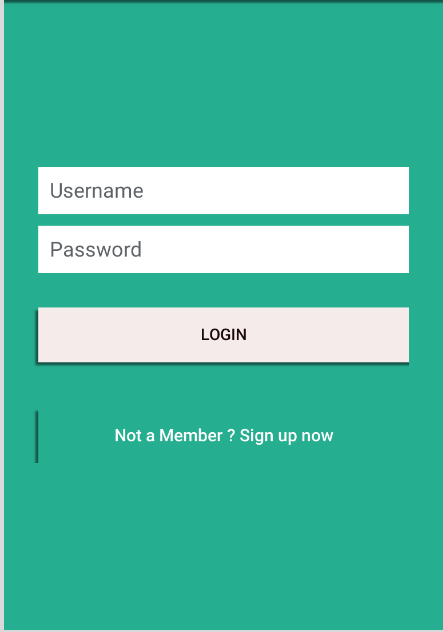
Following is a list of project folders that you would normally have



**Exercise**

Design a neat looking Login and Registration screen that fits all screen sizes.

You can use your own color combination

1. Place three buttons side by side using relative layout

