**Android Basic User Interface: Layouts Overview**

Table of Contents

[View and ViewGroup Objects 2](#_Toc100898224)

[XML Files 3](#_Toc100898225)

[ViewGroup Object Types 4](#_Toc100898226)

[Layout Attributes and Parameters 6](#_Toc100898227)

[Size 8](#_Toc100898228)

[Margins and Padding 9](#_Toc100898229)

[Linear Layout 10](#_Toc100898230)

[Relative Layout 11](#_Toc100898231)

[Constraint Layout 12](#_Toc100898232)

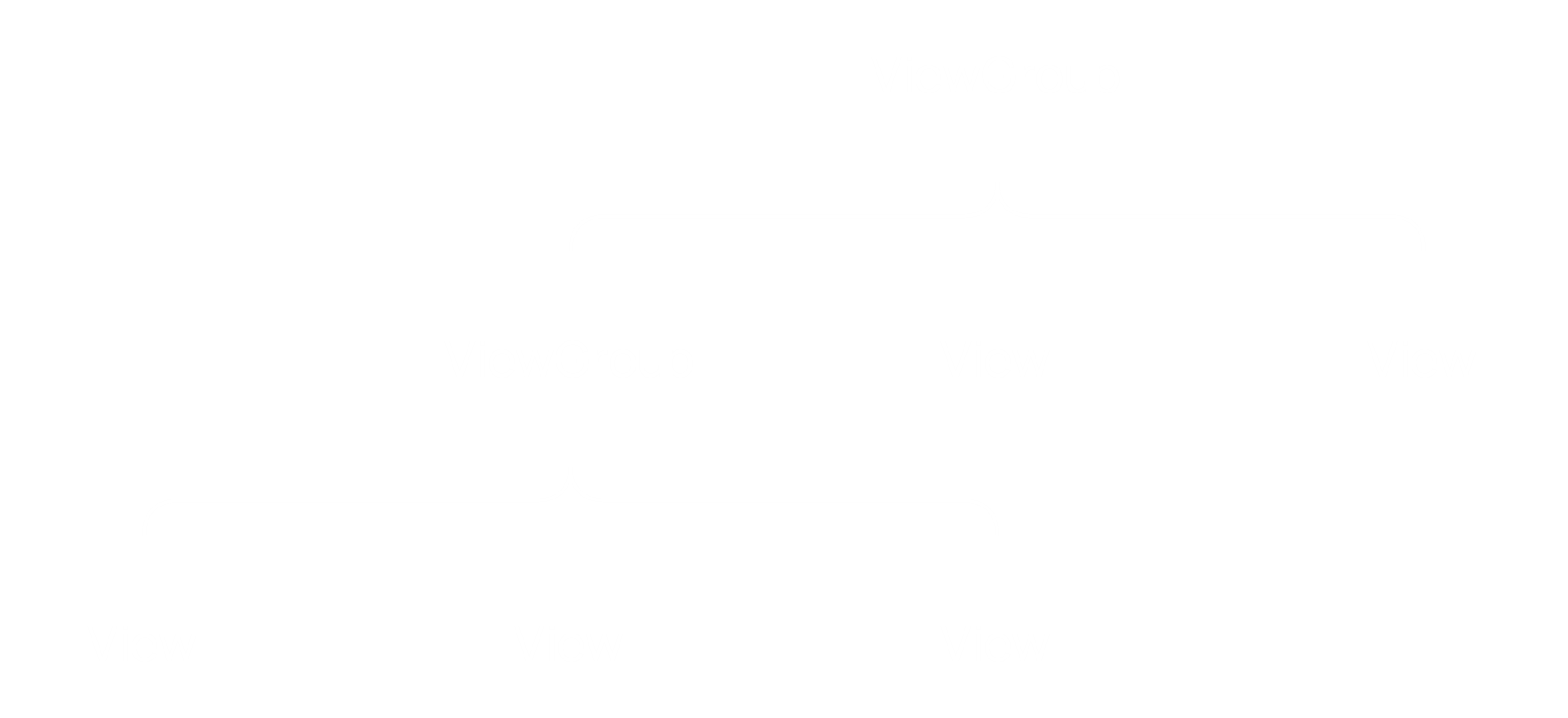
[Differences with Relative Layouts 13](#_Toc100898233)

## View and ViewGroup Objects

A **layout** defines the structure for the user interface in an application, such as in an activity. All user interface elements in an Android application are built using View and ViewGroup objects.

A **View** object draws something onto the screen that the user can interact with. For example, buttons and text fields are View objects.

A **ViewGroup** object is an invisible container that holds other View and ViewGroup objects in a specified order to define the layout of the interface. We will look into the types of ViewGroup objects soon.



From the tree above, we can see that the root-level element is always a ViewGroup object, and we nest View objects and other ViewGroup objects underneath this element. The ViewGroup object holding other View or ViewGroup objects can be considered the parent to the child objects.

## XML Files

Generally, we drag and drop ViewGroup and View objects onto the activity’s layout editor. However, underneath this design editor is an **XML file**.



<ConstraintLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <Button  
 android:id="@+id/button2"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="This is a button"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />

<EditText  
 android:id="@+id/editTextTextPersonName2"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:inputType="textPersonName"  
 android:text="This is an EditText"  
 app:layout\_constraintBottom\_toTopOf="@+id/button2"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
</ConstraintLayout>

XML

In the example above, we have a RelativeLayout, which is a ViewGroup object, containing two View objects, an EditText and a Button. All of these and their corresponding attributes and parameters, which define how they will be positioned and what they will look like, can be found in the XML file. When we drag and drop the elements onto the design editor, this XML file is automatically updated.

## ViewGroup Object Types

There are many different types of ViewGroup objects:

* Linear Layout: A view group that aligns all children in a single direction, vertically or horizontally.
* Relative Layout: A view group that displays child views in relative positions.
* Constraint Layout: A view group where child views can be positioned in adaptable and flexible ways.
* Table Layout: A view group that groups views into rows and columns.
* Absolute Layout: A view group enables you to specify the exact location of its children. This layout is never used.
* Frame Layout: The FrameLayout is a placeholder on screen that you can use to display a single view.
* Motion Layout: A view group that manages view motion and widget animations.
* Coordinator Layout: A view group that enables views to inherit the attributes of the underlying view.
* List View: A view group that displays a list of scrollable items.
* Grid View: A view group that displays items in a two-dimensional, scrollable grid.
* Adapter View: A view group that displays items loaded into an adapter, which is a type of array.

The most common amongst these are the Linear Layout, the Relative Layout and the Constraint Layout. We will be looking into these in more detail soon.

## Layout Attributes and Parameters

**Attributes** define the properties of ViewGroup objects. For example, the width, height, size, etc. of a ViewGroup object are all its attributes.

**Parameters** are properties for View objects. For example, the width, height, size, etc. of a View object are all its parameters.

The attributes that a ViewGroup object has depends on the type of ViewGroup object it is. The parameters that a View object can use are defined by the ViewGroup object under which it resides. There are some common parameters, such as width and height, which apply to nearly every type of ViewGroup object, so every View object can be expected to be able to use those. However, there are also special parameters, such as layout\_alignParentTop, which can only be used by a View object if it happens to be residing under a specific ViewGroup object, a Relative Layout in this case.

Note that the same property can be considered an attribute or a property, depending on whether it is being applied to a ViewGroup object or a View object. For example, layout\_width applies to both. The difference between an attribute and a property is thus (mostly) theoretical.

Properties that are used by a child object are also considered to be the attributes of the parent ViewGroup object, even if those properties are not used by the ViewGroup object itself. For example, the weight property is a parameter for child View objects for a Linear Layout. It is not used by the linear layout itself, but it is still considered one of its attributes, since it allows its children to use the property.

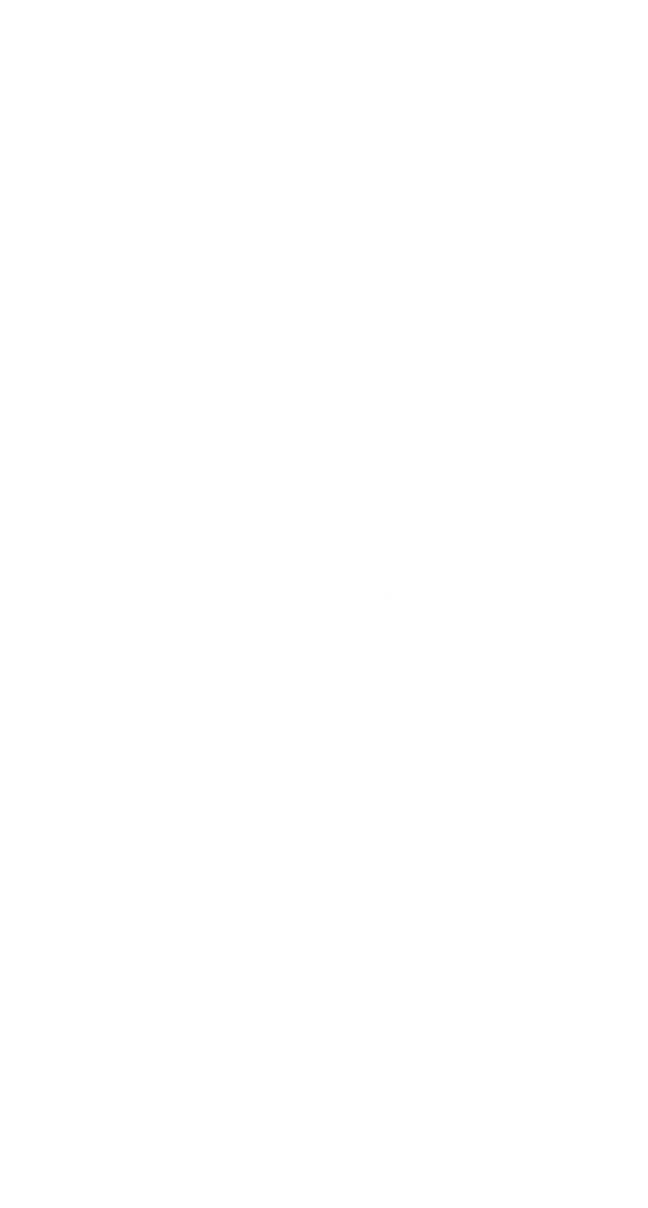
Some common attributes and parameters are:

* id - This is the ID which uniquely identifies the View object inside its parent ViewGroup object.
* layout\_width - This is the width of the layout.
* layout\_height - This is the height of the layout
* layout\_marginTop - This is the extra space on the top side of the layout.
* layout\_marginBottom - This is the extra space on the bottom side of the layout.
* layout\_marginLeft - This is the extra space on the left side of the layout.
* layout\_marginRight - This is the extra space on the right side of the layout.
* layout\_gravity - This is an attribute that specifies how child View objects are positioned.
* layout\_weight - This specifies how much of the extra space in the layout should be allocated to the View object.

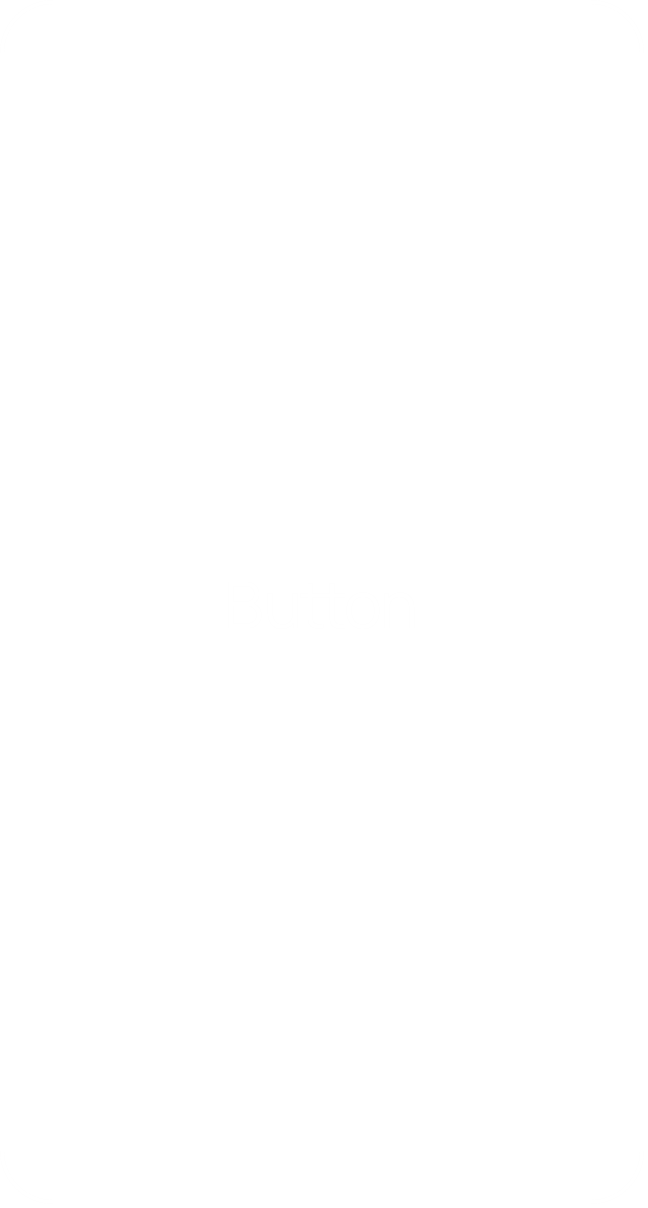
### Size

The layout\_width property, the layout\_height property and many other properties take **sizes** as arguments. Sizes can be of three types:

* A specific, hard-coded value in dp (density independent pixels), sp (scale indepdent pixel), etc.
* wrap\_content, which tells the object to re-size itself to fit its content exactly.



* match\_parent, which tells the object to re-size itself to fill its parent.



Thus, for example, we can set layout\_width="match\_parent" and the object will fill its parent ViewGroup object’s width.

### Margins and Padding

We can set **margins** using the layout\_margin property. This puts extra space around the border of the object.

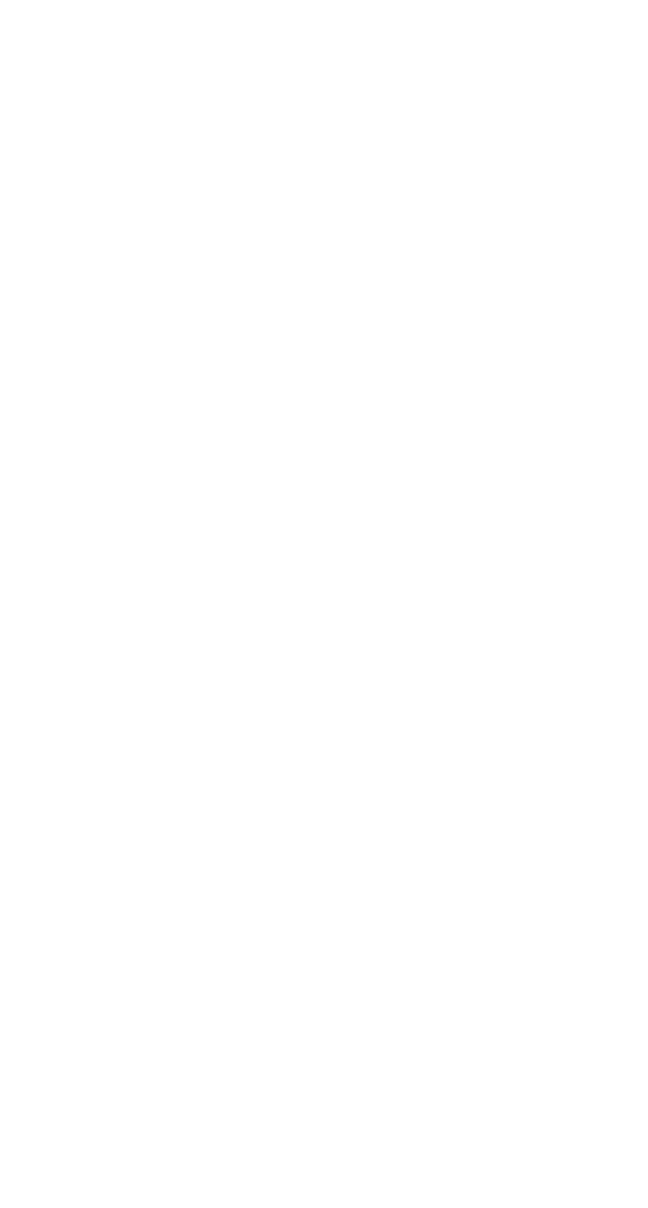
We can set **padding** using the padding property. This puts extra space between the object’s borders and the object’s content.

## Linear Layout

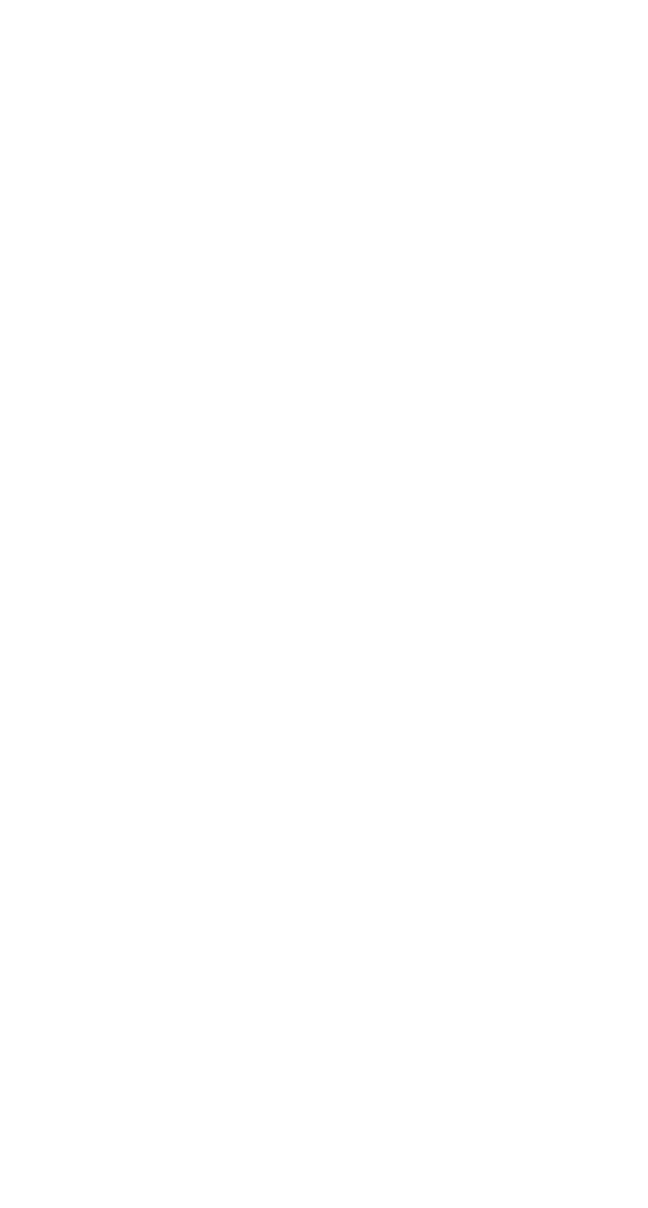
**Linear layouts** can behave either horizontally, placing its child View and ViewGroup objects side-by-side, or vertically, placing its child View and ViewGroup objects beneath each other.

The **attributes** of a linear layout include:

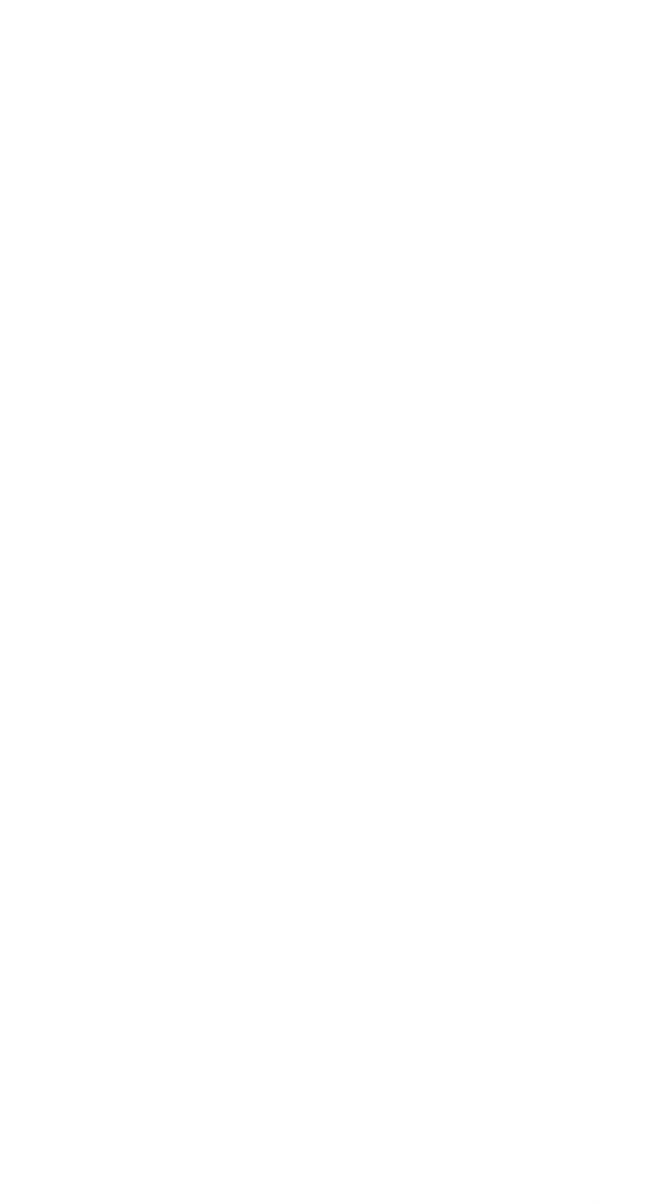
* orientation – This can be either horizontal or vertical.
* layout\_width, layout\_height
* weight – This is a property used by the child objects to define how much of the extra space in the layout will be allocated to that object.



Default Weights



Weights – 1, 1, 0

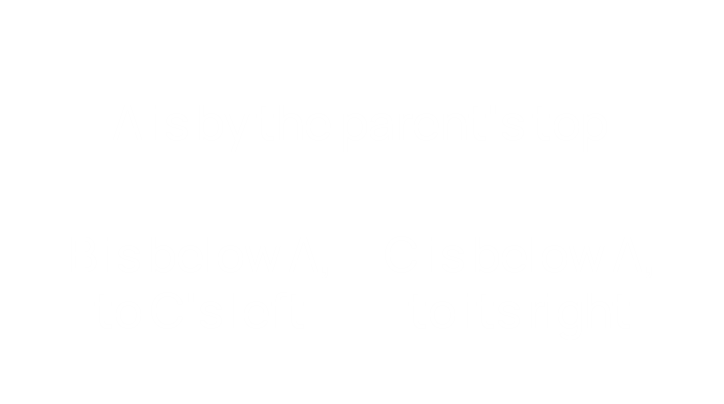


Weights – 1, 1, 2

* gravity – This defines where in the layout the child objects should be placed, top, bottom, centre, left, right, etc.
* padding
* margin

## Relative Layout

In a **relative layout**, the View objects are placed at positions that are relative to other View objects in the layout or to the parent ViewGroup object.



In the example above, the object A is placed at the top of the parent container, the object C is placed below A, to the right of the parent container and the object B is placed below B, to the left of C.

The **attributes** of a relative layout include:

* layout\_alignParentTop, layout\_alignParentBottom, layout\_alignParentLeft, layout\_alignParentRight – If set to true, the View object will be placed along the corresponding edge of the container.
* layout\_centerInParent – If set to true, the View object will be centred both vertically and horizontally in the parent container.
* layout\_centerHorizontal, layout\_centerVertical – If set to true, the View object will be centred horizontally or vertically in the parent container respectively.
* layout\_above, layout\_below, layout\_toLeftOf, layout\_toRightOf – If set to true, the View object will be placed above, below, to the left of or to the right of another specified View object inside the same container respectively.
* layout\_toStartOf, layout\_toEndOf – If set to true, the View object will be placed to the start or end of another specified View object inside the same container respectively. This is different from the previous set of attributes since in some cases, the start might not be the top or the left and the end might not be the bottom or the right.

## Constraint Layout

The **constraint layout** is similar to the relative layout in the sense that we are again placing View object with respect to the position of other View objects. However, the constraint layout is more flexible and easier to use in the layout editor of Android Studio.

In a constraint layout, the distance of the edges of a View object from other View objects or the parent container is specified. For example, we can specify that the left edge should be 10px from another View object’s right edge. We must specify at least one horizontal (either left or right edge) and one vertical (either top or bottom edge) constraint for every View object.

Note however, that we cannot place other **ViewGroup objects** inside a constraint layout.

### Differences with Relative Layouts

* Constraint layouts do not allow **nested ViewGroup objects**. It is said to have a **flat-view hierarchy**.
* Having a flat-view hierarchy gives constraint layouts **better performance**, since it only needs to look out for changes in size for its immediate children, unlike the relative layouts, which need to consider the changes in size for nested children as well. Essentially, constraint layouts never need to **re-compute** sizes.
* We can place View objects inside a constraint layout by using **drag and drop** in the layout editor, whereas for relative layouts, we frequently have to hard code things to get it to work exactly the way we want.