2.1: Linear layout using the Layout Editor

* A ViewGroup is a view that can contain other views. LinearLayout and ScrollView are view groups.
* LinearLayout is a view group that arranges its child views horizontally or vertically.
* Use a ScrollView when you need to display content on the screen, such as long text or a collection of images. A scroll view can contain only one child view. If you want to scroll more than one view, then add a ViewGroup such as a LinearLayout to the ScrollView, and put the views to be scrolled inside that ViewGroup.
* The Layout Editor is a visual design editor inside Android Studio. You can use the Layout Editor to build your app's layout by dragging UI elements into the layout.
* A style is a collection of attributes that specify the appearance for a view. For example, a style can specify font color, font size, background color, padding, and margin.
* You can extract and collect all the formatting of a view into a style. To give your app a consistent look, reuse the style for other views.

2.2: Add user interactivity

* The Layout Editor tool in Android Studio is a visual design editor. You can use the Layout Editor to build your app's layout by dragging UI elements into your layout.
* EditText is a UI element that lets the user enter and modify text.
* A Button is a UI element that the user can tap to perform an action. A button can consist of text, an icon, or both text and an icon.

Click listeners

* You can make any View respond to being tapped by adding a click listener to it.
* The function that defines the click listener receives the View that is clicked.

You can attach a click-listener function to a View in either of two ways:

* In the XML layout, add the android:onClick attribute to the <*View*> element.
* Programmatically, use the setOnClickListener(View.OnClickListener) function in the corresponding Activity.

2.3: Constraint layout using the Layout Editor

* A ConstraintLayout is a ViewGroup that allows you to position and size the layout's child views in a flexible way.
* In a constraint layout, each view's position is defined using at least one horizontal constraint, and at least one vertical constraint*.*
* A constraint connects or aligns a view to another UI element, to the parent layout, or to an invisible guideline.

Advantages of using ConstraintLayout:

* You can make a constraint layout responsive to devices that have different screen sizes and resolutions.
* ConstraintLayout usually results in a flatter view hierarchy than LinearLayout.
* The design editor and the view inspector in Android Studio help you add and configure constraints.

Chains:

* A chain is a group of views that are linked to each other with bidirectional constraints.
* The views within a chain can be distributed either vertically or horizontally.

Design-time attributes:

* Design-time attributes are used and applied only during the layout design, not at runtime. When you run the app, design-time attributes are ignored.
* Design-time attributes are prefixed with the tools namespace. For example, the tools:layout\_editor\_absoluteY and tools:text attributes are design-time attributes.

Baseline constraints:

* A baseline constraint aligns a view's text baseline to the text baseline of another view that has text.
* Baseline constraints are helpful when views have different font sizes.

2.4: Data-binding basics

Steps to use data binding to replace calls to findViewById():

1. Enable data binding in the android section of the build.gradle file:  
   dataBinding { enabled = true }
2. Use <layout> as the root view in your XML layout.
3. Define a binding variable:  
   private lateinit var binding: ActivityMainBinding
4. Create a binding object in MainActivity, replacing setContentView:  
   binding = DataBindingUtil.setContentView(this, R.layout.activity\_main)
5. Replace calls to findViewById() with references to the view in the binding object. For example:  
   findViewById<Button>(R.id.done\_button) ⇒ binding.doneButton  
   (In the example, the name of the view is generated camel case from the view's id in the XML.)

Steps for binding views to data:

1. Create a data class for your data.
2. Add a <data> block inside the <layout> tag.
3. Define a <variable> with a name, and a type that is the data class.

<data>

<variable

name=”myName”

type=”com.example.android.aboutme.MyName”/>

<data>

1. In MainActivity, create a variable with an instance of the data class. For example:  
   private val myName: MyName = MyName("Aleks Haecky")
2. In the binding object, set the variable to the variable you just created:  
   binding.myName = myName
3. In the XML, set the content of the view to the variable that you defined in the <data> block. Use dot notation to access the data inside the data class.  
   android:text="@={myName.name}"