# KCL Tech: Build X: Android – UXml

Don't worry, UXml isn't one more language you have to learn. It's just a tortured pun for what we will be learning today. We will begin to use 'Views' (and 'ViewGroups') to build a simple user interface and then implement some basic functionality in Java.

Help: When you open Android Studio, your code from last week may open up automatically. If so, you'll need to do File > Close Project.

**Recap Challenges:**

For a quick recap we've got a few challenges that use the skills we learnt last week. If you missed it, then you can find the worksheet on GitHub (goo.gl/bj9Khf).

a) Create and run a new Empty Android Project. I will call mine UXml.
b) Add (at least) one more TextView to the layout.
c) Give each TextView a different id.
d) Set the text for each TextView to a different Activity* Life Cycle method. This can be done in either XML or Java. **Hint:** Your Java code contains one.

**Challenge 1: Create user interface using a LinearLayout**

We talked about Views last week. Views are used for displaying content onto the screen such as TextView or can be used to take user input like an EditText. However to make a user interface it is important to be able to combine different Views and control how they look on screen. This is where ViewGroups come in. ViewGroups are a subclass of View (which means they are also Views) however are invisible and are only used to organise and control the layout of all the other views.

The most commonly used ViewGroups are *LinearLayout* and *RelativeLayout*. A LinearLayout will stack Views, one next to each other, either horizontally or vertically. A RelativeLayout allows you to display items on the screen relative to each other, providing more flexibility and freedom over how your layout appears compared to the LinearLayout, but will require more lines of XML.

When the layout for the Empty Activity is generated, it originally is call between two RelativeLayout tags i.e. <*RelativeLayout*> <**/***RelativeLayout*>. However for this challenge, we will be first using a LinearLayout. To change from one to the other, simply replace the tag names (to <LinearLayout>). However LinearLayouts require an additional value: android:orientation which can either be set to *vertical* or *horizontal.* Then just place the Views in order of which you want to come first.

Remember: Closing tags always have a forward slash before the tag name.

The layout that you want to make is the first TextView displaying the app name at the top, a button saying ‘Save’ at the bottom and the space in between split between the second TextView and the EditText. You can control how much of the screen a view takes using android:layout_weight which can be set to a ratio i.e. 1 on both is 1:1.

**Challenge 2: Create the same layout using a RelativeLayout**

a) Create a new Layout file
b) Create the same layout

While we want to create the layout, it would be a good idea to keep both so we can compare the two and look bad at it for future reference. Which is why we will create a new layout file. We can do this by right-clicking on the ‘app’ folder in the Android sidebar. Then New > Layout resource file. Give the layout any name e.g. *layout_main_relative.xml*.

Then we can copy all the views code across or drag and drop new elements. Some useful attributes for RelativeLayout are:

| | | |
|---|---|---|
| android:layout_alignParentTop / android:layout_alignParentBottom | true, false | If "true", makes the top/bottom edge of the view match the top edge of the parent. |
| android:layout_below / android:layout_above | @id/view_id | Positions the top/bottom edge of the view below the view specified with a view ID. |
| android:layout_toRightOf / android:layout_toLeftOf | @id/view_id | Positions the left/right edge of the view to the right/left of the view specified with a view ID. |

**Challenge 3: Append the text from the EditText to the TextView when the Button is pressed**

a) Find EditText and Button in Java
b) Make code run when button is clicked

```
button.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){


        }
});
```

c) Get text from EditText and set it to the text of the TextView

**Extension Challenge 4: Stylise!**

a) Set a background colour/image: android:background=”@color/background_colour”
b) Add an <ImageView />
c) Get inventive

<u>If you require any help, please raise your hand or give us a shout ☺</u>