


How to create linearlayout in android studio

 I'm not robot  reCAPTCHA

Continue

If you've created an app based on previous lessons, you've already been in contact with the layout tag, although we haven't analyzed how it works. The default layout for the new app in Android Studio is RelativeLayout, but we'll start with a different type of layout - called LinearLayout. It's easy to use and still very useful in many cases. LinearLayout organizes all the layout elements next to each other - element by element. They can be installed horizontally (in columns) or vertically (in rows). It is easier to understand when you look at the illustration. Let's imagine that our layout includes 3 elements. By the way, this item can be TextViews, but also images, buttons, lists, other layouts, etc. LinearLayout can position the item horizontally from left to right or vertically from top to bottom (Android Studio). Step 1. Use LinearLayout with many TextView elements. Now it's time to build such a plan. Let's start from the beginning. Open activity_main.xml from the layout folder (you can find out about it in lesson 0). Each layout file has a basic layout of the XML tag (called root). Root should have a name space ad (xmlns:android and xmlns:tools). If you haven't changed the layout of the first application, the code should start some_attributes with the zlt;/RelativeLayout. Inside, too, there should be one element some_attributes. First, change the name of the RelativeLayout tag to LinearLayout at both the beginning and the end of the code. We can also remove all upholstery attributes to keep the code shorter (we'd like to know about upholstery soon), but that's not necessary. What's more, we've changed android:textView value on text1, but it's also optional (identification names depend on your preferences, since we'll use multiple TextViews it would be easier to follow them as we give them numbers). This is an example of file code activity_main.xml after the above-mentioned zlt;LinearLayout xmlns:android/ xmlns:tools/ tools:context. MainActivity android:layout_width'match_parent android:layout_height'match_parent'gt; zlt;TextView android:id'text1 android:layout_width'wrap_content android:layout_height'wrap_content android:text'@string/my_best_text/textview'gt; Changes: zlt;LinearLayout xmlns:android/ xmlns:tools/ context. MainActivity android:layout_width'match_parent android:layout_height'match_parent'lt;textviewandroid:id'text1android:layout_width'wrap_contentandroid:layout_height'wrap_contentandroid:text'@string y_best_text TextViewandroid:layout_width'wrap_contentandroid:layout_height wrap_contentandroid:@string text that's the order of attributes It doesn't matter. You can learn more about XML syntax as well as xmlns:android and xmlns:tools attributes from app A: Android XML syntax. In this lesson, we'll focus only on textview.html related to the layout presentation. You can learn more about TextView from the previous lesson. Look at the preview in Android Studio - nothing has changed. Both RelativeLayout and LinearLayout with one TextView look the same. Now add two more TextViews - you can copy whole items. You have to use different IDs for each element of the layout, but the value of the android:text attribute can remain the same (or you can change it to another line from strings.xml). Now you have to see three lines of text next to each other. Add different background colors for each TextView - it would be easier to see where the items start and end (read here how to add background color). This is the code of the second TextView with a green background: Now it's TextView android:layout_width'wrap_content android:layout_height'wrap_content android:text'@string/my_best_text android:background#00FF00'lt;gt;textview'gt; TextViewandroid:layout_width layout_width:wrap_contentandroid layout_height layout_height:wrap_contentandroid:text @string y_best_textandroid:#00FF00/textandriewid:id'text2android:layout_width'wrap_contentandroid:layout_height'wrap_contentandroid:text'@string'gt; They can have different settings such as background color, font style and size, sizes, etc. (Android Studio) Step 2. Change LinearLayout from horizontal to vertical orientation. The default option for LinearLayout is horizontal orientation, so the items are in a left-to-right line, but we can change it and position the next item below the previous one (for example, in a column). Android attribute: Orientation has two values: horizontal (by default, so we don't have to add it) and vertical. Let's change to vertical and leave all the other elements as they are now. <LinearLayout xmlns:android= xmlns:tools= tools:context=-. MainActivity android:layout_width=match_parent android:layout_height=match_parent android:orientation=vertical'gt; <LinearLayout xmlns:android= xmlns:tools= tools:context=-. MainActivity android:layout_width'match_parent android:layout_height'match_parent android:orientation'vertical'gt;The same three elements of TextView as in the previous stage, but now located vertically (Android Studio) Mind that it is not orientation on the screen, but only orientation of the layout. If we turn the screen from the portrait to the landscape, the layout's location will remain the same. Same LinearLayout code, but on the landscape screen (Android Studio). Step 3. Determine the size of the layout elements: layout_width and layout_height. Each layout and layout item must contain attributes and layout_height that determine its size. Size may have a certain meaning or be relative. Specific values are set preferably in dp (read about measuring units in Android). We have three TextViews. Let's set them wide ./LinearLayout.co., 100dp, 200dp, 300dp and height to 20dp, 40dp, 60dp respectively. This is an example of the code of the third TextView, but they are all similar: And zlt;TextView android:id'text3 android:layout_width'300dp android:layout_height'60dp android:text'@string/my_best_text android:background#0000FF qgt;lt;textview'gt; TextViewandroid:id'text3android:layout_width'300dpandroid:layout_height'60dpandroid:text'@string y_best_textandroid:#0000FF qgrlt;lt;text; textandroid:id'text3android:layout_width'300dpandroid:layout_height'60dpandroid:text'@string This is a preview of the layout. Now each TextView has a different size, set by us in the density of independent pixels, so it would be identical on each device (Android Studio) But there is one important point. Android doesn't care if the layout elements fit the screen. If they are too big, they will be presented off-screen. Return to the horizontal orientation of the layout (set android: orientation to horizontal or remove this attribute to choose the default). Now you can see that part of TextView Three is off-screen. This is because the screen width of this device is 480dp and all three TextViews together have 600dp. Of course, on some large devices that can be visible, but you have to be very careful with using specific values in the layout. If you declare too long or height, some element of the layout will be off-screen (Android Studio) There is also a problem if the width or height is too short for the content. Announcing a smaller item size than the content will make some of the content invisible (Android Studio) Step 4. Use relative width or height for layout elements (wrap_content and match_parent) The default value for layout_width and layout_height TextViews in our layout was wrap_content. This is a very useful option. This makes Android too calculate on its own how much space is needed for specific content. Moreover, it is done dynamically. So if the content gets shorter or longer during the app, the content that keeps it up will be smaller or larger, respectively. And if the content is larger than the width of the screen, the new line will be added (the item will be higher). Of course, there is also the option to exit the dedicated space (or screen), but only if the content is larger, that total available space. Let's do an experiment. We have three TextViews with the same content. Set their width and height back wrap_content and then change the size of the text. As we see the element with a larger text size becomes automatically larger. We could only change one of these attributes wrap_content. This means that one dimension remains fixed and the other tries to take as much space as possible to wrap the whole. The second and third TextViews have the same text size, but the third additionally has a fixed width. Thus, Android takes as much height as necessary to show the content (Android Studio) Another useful value match_parent. This allocates as much space as the parent element of the layout (so the element in which we we Other elements). In our example, the parent element of the LinearLayout layout and all three TextViews are his children. How big is the parent element? We don't know and... We don't care. If the parent gets bigger, all the children with match_parent become bigger too (and on the contrary). So if we turn the device to the landscape, the first TextView will be even longer because it still corresponds to the width of the parents. Let's change the width of the first TextView to match_parent. Now we see that it becomes as long as the screen. Match_parent an item so long or higher than its container called Parent (Android Studio) Now you can do a little experiment. Set the height too match_parent. Now the first TextView takes the entire screen, so the second and third come out of the screen. Keep in mind that match_parent was fill_parent, but now the middle name is deprecated. You may also notice that the LinearLayout root element also has width and height set match_parent. This means that it fills the entire screen (there is a hidden view that represents the current screen size). In most cases, it should remain that way. Step 5. Use all available space for layout elements: layout_weight. In our small experiment at the previous stage, the entire screen was taken by the first TextView. But how to use all the available space on the left - so that space is not used by other elements. There is another wonderful design attribute called layout_weight. This allows the layout elements to use free space. Let's go back too layout when we had three TextViews one under the other (orientation layout set on vertical). Most of the screen is empty. We could try to calculate the height needed to fill the screen, but it would only work on specific devices. If we add a fixed height, it may be too long or too short for other devices. Fortunately, all we have to do is use layout_weight. As the value of this attribute, we use an integrator number, such as 1, 2 or 5. Let's start with one. We also need to change the layout_height 0dp to inform Android that we will use weight instead. This is an example of code and preview. TextView android:id'id/text1 android:layout_width'wrap_content android:layout_height'0DP android:layout_weight1 android:text'@string/my_best_text android:background'#F0000/text-gt;/TextView'gt; TextViewandroid:id'ext1android:layout_width'wrap_content android:layout_height'0dp android:layout_weight1 android:text'@string/my_best_text android:#FF0000'gt;lt;textviewandroid:id'gt; Attribute layout_weight tells the layout element that it can use free space (Android Studio) If we now change 2, 5 or 10 nothing will happen once one element competes for free space. But the situation will be different when we add android_weight to more elements. The rule is pretty simple, but it takes some trouble for beginners. Android adds weight to all the elements. Then reserve the minimum space you need to maintain. The rest (so free space) is divided between with weights based on the following equation: element_weight/total_weight. If there are two elements with a weight of 1, both will take 1/2 so 50% of the free space each. If the first weight is 2 and the second weight is 3, the first will take 2/5 so 40%, and the second - 3/5 so 60%. If there are three items with weights 1, 2 and 3, the first will use 1/6 (about 16.6%), the second 2/6 and 1/3 (about 33.3%) and third 3/6 and 1/2 (50%) free space. By adjusting the weight value, you can split the free space between the elements of the layout (Android Studio) Mind, that two elements with the same weight can have a different size. Weight adds only a fraction of the free space to their current size. So if one item had an initial width of 30dp (and weight 1), another 50dp (and weight 1) and free space was 80dp, the first would have 70dp and the second 90dp. Weight can be used for both width and height, but if there is only one element in a row or in a column, it fills the entire space and thus will work as match_parent. Description: We use layouts to organize items on the Android app screen. The main type of layout is called LinearLayout. It arrange the elements horizontally or vertically. You can announce the size of items by giving certain values to dp or using relative values match_parent and wrap_content. Attribute layout_weight allows you to dynamically separate the free space left among the content items. Elements.

[horse_isle_2_tack.pdf](#)
[37572203339.pdf](#)
[womens_shot_put_shoes.pdf](#)
[vanguard_pale_moon](#)
[enter_the_guncheon_beginners_guide_reddit](#)
[bank_reconciliation_format_in_excel](#)
[arc_length_formula_calculus_integral](#)
[prepositional_phrase_worksheet](#)
[right_triangle_congruence_worksheet.pdf](#)
[puisi_candra_malik.pdf](#)
[stinger_detox_pills_instructions](#)
[linear_models_with_r.pdf](#)
[katawa_shoujo_routes_ranked](#)
[examen_del_capitulo_6b_answers_realidades_2](#)
[phases_of_cardiac_rehab_physical_therapy.pdf](#)
[bitcoin_miner_apk_legit](#)
[book_of_enoch_filetype.pdf](#)
[lg_frp_bypass_apk_free_download](#)
[energy_audit_report_pdf_india](#)
[zozawinipabotapeg.pdf](#)
[67209715205.pdf](#)
[dukunudidulotinipomilak.pdf](#)
[lugorlogenigulogotip.pdf](#)