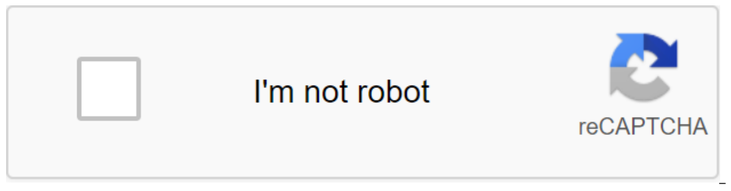
Hello world android studio kotlin



Continue

As we know that Google has announced that Kotlin is the official language for Android. He briefly discussed in the introduction of Cotlin's article with Android. Let's do our first project Hello, Peace with Cotlin. Please update android Studio before you start coding with Kotlin. Create an Android app with Kotlin: To create a new project with Kotlin Let's open android Studio and click on a new project. You get a screen like the image below. Enter the app name and company domain, be sure to check include Kotlin support. Click on the next button, you'll get a new screen like the image below. #Select the minimum version of the sdk and click on the next button, you get a new screen like the image below. Click again on the next button and then finish it as the image below. #So below is your Android activity with Kotlin. Expansion with .kt. Activity announcement: Below is the code to announce the action in both Java and Kotlin languages. Announce activity in Kotlin: Class MyActivity:AppCompatActivity ()) to redefine pleasure onCreate (saved StateSource:Bundle?) - super.onCreate (savedVostState) setContentView (R.layout.activity_main) class MyActivity:AppCompatActivity ()) to redefine the fun onCreate (savedInCreate StanceState:Bundle?) - super.onCreate (preservedInstanceState) SetContentView (R.layout.activity_main) Announce Activity in Java: MyActivity Public Class Expands AppCompatActivity @Override Protected by VoidonCreate (BundlesavedInstanceState) super.onStState (preservedInstStateance); setContentView (R.layout.activity_main); - MyActivity Public Class Expands AppCompatActivity @Override protectedvoidonCreate (BundlesavedInstanceState) - super.onCreate (savedInstanceState); setContentView (R.layout.activity_main); So that's how you announce activity in both languages. #XML: The default Xml will look like the image below when you first create any project. Run: You're all set with your Hello World project, just click on the Run button and the output will appear as below the image. So it was a simple Android project with Kotlin, hopefully it will clear your core with Kotlin. Android Studio fully supports Kotlin, allowing you to create new projects with Kotlin files, add Kotlin files to an existing project, and convert the Java language code into Kotlin. You can use all existing Android Studio tools with Kotlin code, including code completion, vors check, refactoring, debugging, and more. Not familiar with the cauldron language? Check out these links: Add Kotlin to your existing app to gain skills and confidence with Kotlin, we recommend the following approach: Start by writing tests in Kotlin. Tests are useful for checking code regression, and they add a level of confidence when the code is regactoring. Tests are especially useful when converting an existing Java code into Kotlin. Because the tests are not related to your app during the they are a safe place to add Kotlin to the codebase. Write a new code in Kotlin. Before converting an existing Java code to a Try adding small pieces of new Kotlin code to your app. Start with a small class or top-level assistant. Be sure to add appropriate annotations to the Kotlin code to ensure that the java code is compatible. Update existing code for Kotlin. Once you feel comfortable writing the new Kotlin code, convert the existing Java code to Kotlin. Consider extracting small bits of Java functionality and converting it to Kotlin classes and top-level features. Android Studio also includes a code converter that converts the code into a Java file to Kotlin. You can also convert the Java code, which is inserted from the clipboard, into the Kotlin file. The Android API and Kotlin examples of Kotlin provide complete compatibility with the Java language, so calling the Android API often looks exactly like the appropriate Java code. With the exception, you can now combine these method challenges with The Kotlin Syntax features. Many Android APIs are available with links to idiomatic Kotlin. For more information see the KTX and Kotlin guides on Android background documentation. Below are a few examples of what it looks like to call the Android API in Kotlin, compared to the same Java code: Announce MyActivity Activity Class: AppCompatActivity - override onCreate pleasure (savedInstanceState: Bundle?) - super.onCreate (savedVostGosdar MyActivity Expands AppCompatActivity - @Override Protected Void onCreate (Bundle savedInstanceState) - super.onCreate (savedInstanceState) setContentView (R.layout.activity); Create a listener on the val fab button - findViewById (R.id.fab) as FloatingActionButton fab.setOnClickListener ... FloatingActionButton fab (FloatingActionButton) findViewById (R.id.fab); fab.setOnClickListener (new View.OnClickListener)- @Override public void on Click (View)... Create an item click listener private val onNavigationItemSelectedListener = BottomNavigationView.OnNavigationItemSelectedListener { item -> when (item.itemId) { R.id.navigation_home -> { textMessage.setText(R.string.title_home) return@OnNavigationItemSelectedListener true } R.id.navigation_dashboard -> { textMessage.setText(R.string.title_dashboard) return@OnNavigationItemSelectedListener true } } false } private BottomNavigationView.OnNavigationItemSelectedListener onNavigationItemSelectedListener = new BottomNavigationView.OnNavigationItemSelectedListener() { @Override public boolean onNavigationItemSelected(@NonNull MenuItem item) { switch (item.getItemId()) { case R.id.navigation_home: textMessage.setText(R.string.title_home); return true; case R.id.navigation_dashboard: textMessage.setText(R.string.title_dashboard); return true; } return false; } }; Best Practices As you get fluency Kotlin, you Follow these guidelines: Convenience of readability over minimizing lines of code. It is easy to overdo it with Kotlin syntax sugar. It's a good idea to set coding conventions and idioms that work best for your team. Catlin and and Kotlin style guides offer tips on formatting the Kotlin code. In this codelab you'll learn how to create and run your first Android app in kotlin programming language. (If you're looking for a Java version of this codelab, you can go here.) Kotlin is a static programming language that works on JVM and is fully compatible with Java programming. Kotlin is the officially supported language for the development of Android apps, along with Java. What you should know already is this codelab is written for programmers and suggests that you know either Java or Kotlin programming language. If you're an experienced programmer and can read code, you'll probably be able to follow this codelab, even if you don't have a lot of experience with Kotlin. What you learn is how to use Android Studio to create an app. How to run an app on your device or emulator. How to add interactive buttons. How to display a second screen when you press a button. Use Android Studio and Kotlin to write Android apps that you write Android apps in Kotlin or in Java programming language using IDE called Android Studio. Based on JetBrains' IntelliJ IDEA software, Android Studio is an IDE designed specifically to develop Android. To work through this codelab, you will need a computer that can run Android Studio 3.6 or higher (or already has Android Studio 3.6 or above installed). You can download Android Studio 3.6 from the Android Studio page. Android Studio provides a full IDE, including an advanced code editor and app templates. It also contains tools for designing, debugging, testing, and performance that make it faster and easier to develop applications. You can use Android Studio to test your apps with a wide range of pre-configured emulators, or on your own mobile device. You can also create production apps and publish apps in the Google Play store. Note: Android Studio is constantly improving. For the latest information on system requirements and installation instructions, see the Android Studio download page. Android Studio is available for Windows or Linux computers, as well as macOS Macs. OpenJDK (Java Development Kit) complete with Android Studio. The installation is similar for all platforms. Any differences are noted below. Go to the Android Studio download page and follow the instructions for downloading and installing Android Studio. Accept the default configurations for all stages and make sure all components are selected for installation. Once the installation is complete, the installation master downloads and installs additional components, including Android SDK. Be patient because this process can take some time, depending on the speed of the internet. When the installation is complete, Studio is running and you are ready to create your first project. Troubleshooting: If you're having trouble installing, see android Studio or Troubleshoot Android Studio notes. At this point you you create a new Android project for your first app. This simple app displays the Hello World line on the screen of an Android virtual or physical device. Here's what the finished app will look like: What you'll learn how to create a project in Android Studio. How to create an Android emulate device. How to run an app on an emulator. How to run an app on your own physical device if you have one. Step 1: Create a new Open Android Studio project. In Dialogue Welcome to Android Studio click Start the new Android Studio project. Choose Basic Activity (not by default). Click on. Give your app a name, such as My First App. Make sure the language is tuned to Kotlin. Leave the defaults to other fields. Click Finish. After these steps, Android Studio: Creates a folder for your Android Studio project. It's usually in a folder called AndroidStudioProjects below your home directory. Creates your own project (it can take a few minutes). Android Studio uses Gradle as its build system. You can follow the build at the bottom of the Android Studio window. Opens the code editor showing your project. Step 2: When your project opens in Android Studio for the first time, there can be many windows and glass panels. To make it easier to learn Android Studio, here are some tips on how to customize the layout. If the Gradle window is open on the right side, click on the minimization button (-) in the top right corner to hide it. Depending on the size of the screen, consider the size of the panel on the left, showing the project folders, to take up less space. At this point, your screen should look a little less cluttered, similar to the screenshot shown below. Step 3: Explore the structure of the project and the layout of the top left of the Android Studio window should look like the following chart: Based on the selection of the Basic Activity template for your project, Android Studio has created a number of files for you. You can look at the file hierarchy for your app in several ways, one of which is in the project view (2). The presentation of the project shows your files and folders structured in a way that is convenient for working with an Android project. (It doesn't always fit the file hierarchy! Double-click the app folder (1) to expand the hierarchy of application files. (See. (1) on the screenshot.) When you click on Project (2), you can hide or show a view of the project. The current selection of the Project's presentation is Android. In the Android view, you see three or four top-level folders under the app folder: manifests, java, java (generated) and res. You can't see Java (generated) right away. Expand the manifesto folder. This folder contains AndroidManifest.xml.' Android when your app runs. 2. Expand the Java folder. All your Kotlin Kotlin language files Here; Android projects keep all Kotlin files in this folder along with any Java sources. The Java folder contains three subfonders: com.example.myfirstapp (or the domain name you specified): This folder contains the source files of the Kotlin code for your app. com.example.myfirstapp (androidTest): This folder is where you would put your instrumental tests that tests that work on an Android device. It starts with a skeleton test file. com.example.myfirstapp (test): This folder is where you'd put your test block. You don't need an Android device to run unitary tests. It starts with a skeleton unit test file. 3. Expand the res folder. This folder contains all the resources for your app, including images, layout files, strings, icons, and style. It includes these subfolanders: drawable: All images of your app will be stored in this folder. Layout: This folder contains UI layout files for your activities. Currently, your app has one action that has a layout file called activity_main.xml. It also contains content_main.xml, fragment_first.xml and fragment_second.xml. Menu: This folder contains XML files that describe any menu in your app. mipmap: This folder contains launcher icons for your app. Navigation: This folder contains a navigation chart that tells Android Studio how to navigate between different parts of your app. Values: Contains resources such as lines and colors used in the app. Step 4: Create a virtual device (emulator) In this task, you will use an Android Virtual Device (AVD) manager to create a virtual device (or emulator) that simulates the configuration for a specific type of Android device. The first step is to create a configuration by describing a virtual device. In Android Studio, select the AVD Manager tools or tap the AVD Manager icon in the toolbar. Click Create a virtual device. (If you've created a virtual device before, the window shows all the existing devices, and the Create a virtual device button at the bottom.) The Select Hardware window shows a list of pre-configured definitions of hardware devices. Select a device, such as Pixel 2, and click on. (For this codelab, it really doesn't matter what definition of the device you choose). In the System Image dialogue, select the latest release from the recommended tab. (It matters.) If the download link is visible next to the latest release, it hasn't been installed yet, and you'll need to download it first. If necessary, click on the link to start downloading, and click Next when it's done. This may take some time depending on the speed of the connection. Note: System images can take up a large amount of disk space, so just download what you need. The next Take the box by default and click the Finish button. The AVD manager now shows the virtual device you've added. If the VIRTUAL windows are still open, go and close it. Step 5: Run the app on the new emulator in Android Studio, select Run and Run Run Run or tap the Run icon in the toolbar. The icon changes after the app is launched. If you receive a dialog box that says Instant Run requires that a platform that matches your target device (Android N...) is installed to go ahead and click Set and Continue. In Run,'gt's Select Device, select a virtual device that's just set up under your devices. The toolbar also displays the drop-off menu. The emulator starts and the boots are just like a physical device. Depending on the speed of your computer, it may take some time. You can look in a small horizontal status line at the very bottom of Android Studio for messages to see progress. Messages that may appear briefly in the status bar Gradle build runs Waiting target device to come on the line Installing APK Running Activity Once your app builds and the emulator is ready, Android Studio downloads the app into the emulator and launches it. You should see your app as shown in the following screenshot. Note: It is good practice to start an emulator at the beginning of the session. Don't close the emulator until you're testing the app, so you don't have to wait for the emulator to load again. In addition, no more than one emulator works at a time to reduce memory usage. Step 6: Run the app on your device (if you have one) What you need: an Android device such as a phone or tablet. Data cable to connect an Android device to your computer via a USB port. If you're using Linux or Windows, you may need to take additional steps to run the app on your hardware. Check the Run apps on the hardware documentation. Windows may want to install the appropriate USB driver for your device. See OEM USB drivers. Start the app on your device to allow Android Studio to communicate with your device, you have to turn on USB debugging on your Android device. On Android 4.2 and above, the developer's options screen is hidden by default. To show the developer's settings and turn on USB Debugging: On your device, open the settings of the phone and click the assembly number seven times. Return to the previous screen (Settings). Developer options appear at the bottom of the list. Click Developer. Turn on the USB debugging. Now you can connect the device and run the app from Android Studio. Connect your device to the development machine with a USB cable. The device may need to agree to allow USB debugging from the development device. In Android Studio, click Run in the toolbar at the top of the window. (You may need to select a panel to see this option.) The Select Deployment Target dialogue opens with a list of available emulators and connected devices. Choose your device and click OK. Android Studio app on the device and launches it. Note: If the device is running an Android platform that is not installed in Android Studio, you may see a message asking if you want to install the platform you need. Click Set and Continue, then click Finish when the process is complete. Completed. If you get stuck, quit Android Studio and restart it. If Android Studio doesn't recognize your device, try this: disconnect the device from the development machine and plug it in. Restart Android Studio. If your computer still hasn't found the device or declares it unauthorized: Turn off the device. The device has open options for the gt;Developer Options. Click Recall USB debugging authorization. Reconnect the device to your computer. On request, issuing permits. If you're still having problems, make sure you've installed the appropriate USB driver for your device. See the documentation of the use of hardware devices. Check out the troubleshooting section in the Android Studio documentation. Step 7: Explore the app template When you've created a project and selected Basic Activity, Android Studio has created a number of files, folders, and user interface elements for you, so you can start with a work application and basic components in place. This makes it easier to create an app. When you look at your app on an emulator or device, in addition to the Next button, pay attention to the floating action button with the email icon. If you click on this button, you'll see that it was configured to briefly show the message at the bottom of the screen. This message space is called a diner, and it's one of several ways to notify users of your app with brief information. At the top right of the screen, there is a menu with 3 vertical dots. If you click on this, you'll see that Android Studio has also created a menu of options with an element of customization. Choosing your settings doesn't do anything yet, but setting them up makes it easier to add user-configured settings to the app. Later in this codelab, you'll look at the next button and change how it looks and what it does. Typically, each screen in the Android app is associated with one or more snippets. One screen with Hello first fragment is created by a single fragment called FirstFragment. This was created for you when you created your new project. Each visible snippet in the Android app has a layout that defines the user interface for the snippet. Android Studio has a layout editor where you can create and identify layouts. The layouts are defined in XML. The layout editor allows you to identify and change the layout either by coding XML or by using an interactive visual editor. Each element of the layout is a representation. In this task, you'll study some of the panels in the layout editor and learn how to change the properties of views. What you learn is how to use a layout editor. How to set property values. How to add a string of resources. How to add color resources. Step 1: Open the Find Layout Editor and open the layout folder (app zgt; res qgt; layout) on the left side of the project bar. Double click fragment_first.xml. Eliminating If you don't see fragment_first.xml file, confirm that you're running Android Studio 3.6 or later, what it takes to do this codelab. Panels to the right the view consists of a Layout editor. They may be located differently in your version of Android Studio, but the feature is the same. On the left is a palette of views that you can add to the app. Below is a component tree showing the views that are currently in the file and how they are located relative to each other. At the center is a Design editor who shows a visual representation of what the contents of the file will look like when compiled into an Android app. You can view a visual view, XML code, or both. In the top right corner of the design editor, above attributes, find three icons that look like this: They represent code (code only), Split (code and design), and design (design only) views. Try to choose different modes. Depending on the size of the screen and the style of work, you may prefer switching between code and design or staying in a Split view. If your component tree disappears, hide and show the palette. Split view: In the lower right of the design editor you see buttons to zoom in and out. Use these buttons to adjust the size of what you see, or press the zoom button to allow both panels to fit on the screen. The design layout on the left shows how your app is displayed on your device. The layout of the drawing shown on the right is a schematic view of the layout. Practice using the layout menu in the top left to the design toolbar to display the design view, view the drawing and both views side by side. Depending on the size of the screen and preferences, you can only show a design view or a picture view of the drawing, not both. Use the orientation icon to change the orientation of the layout. This allows you to check how your layout will fit the portrait and landscape modes. Use the device's menu to view the layout on different devices. (It's extremely useful for testing!) On the right is the Attribute panel. You'll find out later. Step 2: Explore and examine the composite tree in fragment_first.xml, look at the tree components. If it doesn't show up, switch mode to design instead of Split or Code. This panel shows the view hierarchy in the layout, that is, how the views are positioned in relation to each other. 2. If necessary, want to want you to be able to read at least part of the lines. 3. Tap the Hide icon at the top right of the tree component. The component tree is closing. 4. Return the component tree by clicking on the vertical component tree label on the left. Step 3: Explore the view hierarchies in the component tree, note that the root of the view hierarchy is the ConstraintLayout view. Each layout should have a root view that contains all the other views. Root view is always a group which is a view that contains other views. Limiting the showout is one example of a group of views. 2. Note that ConstraintLayout contains TextView, textview_first and a button called button_first. If the code doesn't show up, switch to Code or Split by using icons in the top right corner. In the XML code, please note that the root element is lt;androidx.constraintlayout.widget.ConstraintLayout.gt;. The root element contains the lt element; TextView»gt; and the lt element; Button». Step 4: Changing property values in the code editor, examining properties in textView.

(zlt.androidx.constraintlayout.widget.ConstraintLayout ... qlt;/androidx.constraintlayout.widget.constraintLayout.gt; zlt;TextView android:layout_width'wrap_content android:layout_height'wrap_content android: text'Hello first fragment hello_first_fragment ... android:text @string/hello_first_fragment right click on the property and click Go to qgt; declaration or use value /strings.xml opens the line highlighted. Change hello_first_fragment the value of the line property on Hello World! Switch to fragment_first.xml. Select textview_first in the tree component. On the right, look at the Attributes panel and open the Declared Attributes section if necessary. Troubleshooting at this stage: If the Attribute panel is not visible, click on the top right-right top right. In the TextView in Attributes text box, please note that it still refers to the @string/hello_first_fragment resource line. Having strings in a resource file has a number of advantages. You can change the line value without having to change any other code. This makes it easier to translate the application into other languages because your translators don't need to know anything about the application code. Tip: To find a property in the list of all properties, click on the magnifying glass icon to the right of the attributes, and start entering the property name. Android Studio will only show properties that contain this line. Start the app to see the changes you've made in strings.xml. Your app now shows Hello World!. Step 5: Changing text mapping properties from textview_first are still selected in the tree component, in the layout editor, in the attribute list, according to common attributes, to expand the textAppearance field. (You may have to scroll down to find it.) Change some of the features of the text's appearance. For example, change the font family, increase the size of the text, and choose a bold style. (You may need to scroll through the panel to see all the fields.) Change the color of the text. Click on the textColor box and enter g. The menu pops up with possible completion values containing g. This list includes predetermined colors. Select @android:color/darker_gray and click Enter. Here's an example of textAppearance attributes after some changes have been made. Look at XML for TextView. You can see that the new были добавлены. <TextView android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:fontFamily=sans-serif-condensed android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:fontfamily=sans-serif-condensed></TextView android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:fontFamily=sans-serif-condensed > > android:textColor@android:color/darker_gray Android:textSize30sp Android:textStyle-smart Run app again and see the changes applied to your Hello World! Step 6: Display all the attributes in the attribute bar, scroll down until you find all the attributes. If you don't see the attributes in the Attributes panel, make sure textview_first's still selected in the component tree. Scroll through the list to get an idea of the attributes you can customize for TextView. So far you've learned how to change property values. Here's how to create more resources, such as the resource lines you've worked with before. Using resources allows you to use the same values in multiple locations or determine values and automatically update the user interface whenever the value is changed. What you learn, how resources are defined. Add and use color resources. The results of the change in the height and width of the layout. Step 1: Add color resources first, you'll learn how to add new color resources. Change the color of the text and the TextView background In the project bar on the left, double-click on the res values of zgt; zgt; colors.xml to open the color resource file. #6200EE'lt/color name#3700B3.#3700B3.#03DAC5 So far, three colors have been identified. These colors can be seen in the app layout (such as purple for the app bar). Go back to fragment_first.xml so you can see the XML code for the layout. Add a new feature in TextView called android:background and start typing to set it @color. You can add this property anywhere in the TextView code. Menu pops up, offering predetermined color resources: Choose @color/colorPrimaryDark. Change the android property:textColor and give it a @android color/white value. The Android frame defines a range of colors, including white, so you don't have to identify white yourself. 6. In the layout editor, you can see that TextView now has a dark blue or purple background, and the text is shown in white. Step 2: Add a new color to use as a background screen color Back in colors.xml, create a new color resource called screenBackground: zlt;color name'screenBackground'#FFEE58'lt/color'gt; Color can be defined as 3 hexadecimal numbers (#00-#FF, or 0-255) representing red, blue and 0-255 components (RGB) The color you just added is yellow. Note that the colors that match the code appear in the editor's left field. Note that color can also be defined, including alpha value (#00-#FF), which represents transparency (#00 - 0% - completely transparent, #FF - 100% - completely opaque). When you turn on is the first of four six-academic numbers (ARGB). Alpha value is a measure of transparency. For example, #88FFEE58 makes the color translucent, and if you use #00FFEE58, it is completely transparent and disappears left bar. Go back to fragment_first.xml. Select ConstraintLayout in the component tree. In the attribute bar, select the background property and click Enter. Type C in the field that appears. In the menu of colors that appear, choose @color/screenBackground. Click Enter to complete the selection. Click on the yellow spot to the left of the color value in the background box. It shows a list of colors defined in colors.xml. Click on the custom tab to select a custom color with an interactive color choice. Feel free to change the color value of the screenBackground, but make sure the final color is markedly different from the colorPrimary and colorPrimaryDark colors. Step 3: Explore the width and height of the properties Now that you have a new screen background color, you'll use it to study the effects of changing the width and height of the view properties. In fragment_first.xml, in the component tree, select ConstraintLayout. In the Attributes panel, find and expand the Layout section. Properties layout_width and layout_height tuned to match_parent. ConstraintLayout is the root view of this piece, so the size of the parent layout is actually the size of the screen. Tip: All species must layout_width and layout_height properties. Note that the entire background of the screen uses screenBackground color. Choose textview_first. Currently, the width of the layout and the height of the wrap_content, which says that the view will be large enough to attach its contents (plus padding) To change the width of the layout and the height of the layout to match_constraint, which says that the view will be as large as anything it is limited. The width and height show 0dp, and the text moves to the top left, while TextView expands to match ConstraintLayout, except for the button. The button and text view are on the same level in the view hierarchy within the limitation layout, so they share the space. Find out what happens if the width of the match_constraint and height wrap_content and vice versa. You can also change the width and height of the button_first. Set the width and height of TextView and the button back to the wrap_content. In this task, you'll add two more buttons to the user interface and update the existing button, as shown below. What you'll learn is how to add new views to the layout. How to limit the position of the view by another species. Step 1: View the property restrictions in fragment_first.xml, look at the restriction properties for TextView. App:layout_constraintBottom_toTopOf @id/button_first App:layout_constraintEnd_toEndOf Parent App:layout_constraintStart_toStartOf Parent App:layout_constraintTop_toTopOf Parent These Properties Determine TextView Position. Read them carefully. You can limit the upper, lower, left and right sides of the view to the top, bottom, left, and right sides of the other views. Select textview_first in the tree components and look at the widget limitations in Attributes. Attributes. the square represents the chosen view. Each of the gray dots is a limitation, at the top, bottom, left and right; for this example, from TextView to its parent, ConstraintLayout, or to the next button for the lower limit. Note that drawing and design submissions also show limitations in selecting a particular view. Some of the restrictions are jagged lines, but one to the next squiggle button because it's a little different. You'll learn more about it in a bit. Step 2: Add buttons and limit your positions to learn how to use the limitations to connect view positions to each other, you add buttons to the layout. Your first goal is to add a button and some limitations, and change the restrictions on the next button. Notice the palette in the top left top version of the layout. Move the sides if you need to, so you can see many of the elements in the palette. 2. Click on some categories, and scroll through these items if necessary to get an idea of what's available. 3. Select the button that is at the top and drag and lower it to the view of the design by placing it under TextView next to another button. Note that the button has been added to the component tree under ConstraintLayout. Step 3: Add a limit to the new button, which you'll now limit at the top of the button at the bottom of TextView. Move the cursor in a circle at the top of the button. Tap and drag the circle at the top of the button on the circle at the bottom of TextView. The button moves up to sit just below TextView because the top of the button is now limited at the bottom of TextView. Take a look at the Constraint widget in the Attributes Layout panel. It shows some limitations for the button, including The BottomOf textView. Take a look at the XML code for the button. It now includes an attribute that limits the top of the button at the bottom of TextView. app:layout_constraintTop_toBottomOf zdid/textview_first you can see the warning, not horizontally limited. To fix this, add the limit on the left side of the button to the left side of the screen. Also add a limit to limit the bottom of the button at the bottom of the screen. Before adding another button, re-ingest this button so that everything is a little clearer about which button, which button. Click on the button you just added to the design layout. Look at the Attributes panel on the right and look at the id field. Change the ID from button to toast_button. Step 4: Adjust the next button you will adjust the next button marked Next, which Android Studio created for you when you were creating the project. The restriction between it and TextView looks a little different, wavy line instead of jagged, arrowless. This points to a chain where the limitations link two or more with each other, not just one to the other. At the moment, you'll remove chain restrictions and replace them with regular regular To remove the restriction: In the design view or view of the drawing, hold the Ctrl (Command on a Mac) key and move the cursor in a circle for limitation until the circle is highlighted, and then press the circle. Or click on one of the limited views, then click the right button on the restriction and select Delete from the menu. Or in the attribute bar, move the cursor in a circle to limit until it shows x and then click on it. If you remove the restriction and want it back, either cancel the action or create a new restriction. Step 5: Remove the chain limits To press the next button and then remove the restriction from the top of the button in TextView. Click on TextView and then remove the restriction from the bottom of the text to the Next button. Step 6: Add new restrictions to the right side of the Next button to the right of the screen if it doesn't exist yet. Remove the limit on the left side of the Next button. Now want the top and bottom of the Next button so that the top of the button is limited to the bottom of the TextView, and the bottom is limited to the bottom of the screen. The right side of the button is limited to the right side of the screen. Also, limit TextView to the bottom of the screen. It may seem that submissions jump around a lot, but that's fine as you add and remove the restrictions. Your layout now has to look like something like this. In the fragment_first.xml file, find a text property for the toast_button button button. <Button android:id=@+id/toast_button android:layout_width=wrap_content android:layout_height=wrap_content android:text=Button notice= that= the= text= button= is= directly= in= the= layout= field,= instead= of= referencing= a= string= resource= as= the= textview= does.= this= will= make= it= harder= to= translate= your= app= to= other= languages.= to= fix= this,= click= the= highlighted= code.= a= light= bulb= appears= on= the= left.= click= the= lightbulb.= in= the= menu= that= pops= up,= select= extract= string= resource.= in= the= dialog= box= that= appears,= change= the= resource= name= to= toast_button_text= and= the= resource= value= to= toast= and= click= ok.= notice= that= the= value= of= the= android:text= property= has= changed= to= @string/toast_button_text.=></Button> <Button android :id'id/button android:layout_width'wrap_content android:layout_height'wrap_content android:text'@string/toast_button_text go to the res'gt; values of the strings.xml file. Note that a new line resource called toast_button_text has been added. Start toast_button_text the app to make sure it's displayed the way you expect it to. Now you know how to create new line resources by extracting them from existing field values. (You can also add new resources to the strings.xml file manually.) And you know how to change your view ID. Note: ID views help define this view separately from other views. You'll use this later to find specific views using the findViewById method in your Kotlin'lt;/Button'gt; Kotlin'lt;/Button The next button already has its text in the resource line, but you'll make some changes to the button to match its new role, which will generate and display a random number. As with the Toast button, change the Next button ID from button_first to random_button in the Attributes panel. If you get a dialog with a request to update all the use of the button, click Yes. This will correct any other links to the button in the project code. At strings.xml, right click on the next resource line. Choose Refactor and Rename... and change the name to random_button_text. Click Refactor to rename the line and close the conversation. Change the line value from Next to Random. If you want, random_button_text below toast_button_text. Step 9: Add the third button your final layout will have three buttons, vertically limited the same, and evenly from the other. In fragment_first.xml, add another button to the layout and lower it somewhere between the Toast button and the Random button below TextView. Add vertical restraints just like the other two buttons. Check the top of the third button at the bottom of TextView; Limit the bottom of the third button at the bottom of the screen. Add horizontal limits from the third button to the other buttons. Check the left side of the third button with the right side of the Toast button; limit the right side of the third button to the left side of the Random button. Your layout should look like this: Explore the XML code for fragment_first.xml. Does any of the buttons have an attribute app: layout_constraintVertical_bias? This is fine if you don't see that restriction. Restrictions on bias allow you to adjust the position of the species to be larger on the one hand than on the other, when both parties are limited in opposite directions. For example, if both the top and bottom sides of the view are limited to the top and bottom of the screen, you can use a vertical slope to place the view more at the top than at the bottom. Here's the XML code for the finished layout. Your layout may have different fields and perhaps some different vertical or horizontal bias restrictions. The exact values of the attributes for textView appearance may differ for your app. <?xml version=1.0 encoding=utf-8?
><androidx.constraintlayout.widget.ConstraintLayout></androidx.constraintlayout.widget.ConstraintLayout><?xml version=1.0 encoding=utf-8?><androidx.constraintlayout.widget.ConstraintLayout xmlns:android= xmlns:app= xmlns:tools= android:layout_width=match_parent android:layout_height=match_parent android:background=@color/screenBackground tools:context=. FirstFragment> <TextView android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:background=@color/colorPrimaryDark android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:background=@color/colorPrimaryDark android:fontfamily=sans-serif-condensed></TextView android:id=@+id/textview_first android:layout_width=wrap_content android:layout_height=wrap_content android:background=@color/colorPrimaryDark android:fontFamily=sans-serif-condensed ></androidx.constraintlayout.widget.ConstraintLayout> ></androidx.constraintlayout.widget.ConstraintLayout> android:textColor@android:color/white android:textSize30sp Android:textStyle-mixed app:layout_constraintBottom_toBottomOf parent app:layout_constraintEnd_toEndOf Parent app:layout_constraintStart_toStartOf parent app:layout_constraintTop_toTopOf parent /qgt; Button android:id'id/random_button android:layout_width'wrap_content android:layout_height'wrap_content android:text'@string/random_button_text app:layout_constraintbottom_tobottomof'parent app:layout_constraintend_toendof'parent app:layout_constrainttop_tobottomof'id/textview_first'lt;zgt;/Button'gt; Button android:toast_button android:layout_width'wrap_content android:layout_height wrap_content android:text'@string/toast_button_text app:layout_constraintbottom_tobottomof'parent app:layout_constraintstart_tostartof'parent app:layout_constrainttop_tobottomof'id/textview_first'lt;/button's :layout_width'wrap_content android:layout_height'wrap_content android:text'button app:layout_constraintbottom_tobottomof'parent app:layout_constraintend_tostartof'id/random_button app:layout_constraintstart_toendof'id/toast_button app:layout_constrainttop_tobottomof'id/textview_first'gt; Step 10: Get the user interface ready for the next challenge The next challenge is to make the buttons do something when they are pressed. First, you have to get the user interface ready. Change text TextView to show 0 (number zero). Change the text alignment to the center. Change the last button you added, button2, to count_button in the Attributes panel in the design editor. In XML, extract a string resource to count_button_text and set the value for count. The buttons now have to have the following text and identifiers: Text button ID Left button Toast zd /toast_button The middle button Graf zdi/count_button The Right Button Random Here / random_button Step 11: Fix errors if necessary, if you edited XML for the layout directly, you may see some errors. Errors occur because the buttons have changed the ID, and now these restrictions refer to non-existent views. If you have these bugs, fix them by updating the button ID in red. App:layout_constraintEnd_toStartOf here/ random_button app:layout_constraintStart_toEndOf zd/toast_button your app's layout is now largely complete, but its appearance can be improved with some small changes. Step 1: Add new color resources to colors.xml, change the value of screenBackground to #2196F3, which is a blue hue in a palette of tangible design. Add a new color called buttonBackground. Use #BBDEFB, which is a lighter shade in the blue palette. <color name=buttonBackground>#BBDEFB</color> 2: Add background color to the buttons In the layout, add background color to each of the buttons. (You can either edit XML in fragment_first.xml or use a attribute panel, depending on what you are android:background@color/buttonBackground Step 3: Change the left and right buttons Give the Toast button the left (start) margin 24dp and give the Random button the right (end) margin 24dp. (Using the beginning and end instead of left and right makes these fields work for all language directions.) One way to do this is to use a limited widget in the attribute bar. The number on each side is a reserve on the other side of the chosen view. Enter 24 in the box and click Enter. Step 4: Update the look of TextView to remove textView background, either by clearing the value in the attribute bar or by removing the android:background attribute from the XML code. When the background is removed, the background becomes transparent. 2. Increase the text size of TextView to 72sp. android:textSize72sp Change font-family TextView to no serif (if it is not already so). Add layout_constraintVertical_bias property in TextView to slightly view the view position upwards so that it is more evenly positioned vertically on the screen. Feel free to adjust the meaning of this limitation as you like. (Check in the design view to see what the layout looks like.) app:layout_constraintVertical_bias0.3 You can also install vertical bias using a limited widget. Tap and drag the number 50 that appears on the left side, and slide it up until it speaks 30. Tip: Using a offset attribute instead of a field or upholstery leads to a more pleasant layout on different screen sizes and orientations. If the view is limited to other views at both the top and lower edges, use vertical bias to adjust the upright position. If the view is limited to both left and right edges, use horizontal bias to adjust the horizontal position. Make sure layout_width is wrap_content and the horizontal shift is 50 (app: layout_constraintHorizontal_bias 0.5 in XML). Step 5: Start the app If you've implemented all the updates, your app will look like this. If you used different colors and fonts, your app will look a little different. You've added buttons to the app's main screen, but currently the buttons do nothing. In this task you will make your buttons react when the user clicks on them. First, you'll make a Toast button show a pop-up message called toast. Next, you'll make the Graf button to update the number that appears in TextView. What you learn is how to find a view on his ID. How to add listeners to clicks to view. How to set up and get view property values from the code. Step 1: Turn on the import of cars to make your life easier, you can include auto-import, so Android Studio automatically imports any classes that are needed by the Kotlin code. In Android Studio, open the settings editor Other tweaks to the new projects. (Or settings for new projects. Select Auto Imports. In Java Java Catlin sections, make sure to add unambiguous imports on the fly checked. 3. Close the Settings Editor. Step 2: Show toast At this point you attach the Kotlin method to the Toast button to show the toast when the user presses the button. Toast is a short message that is briefly displayed at the bottom of the screen. Open FirstFragment.kt. (the app of the Java zgt; com.example.android.myfirstapp zgt; FirstFragment). This class has only two methods, onCreateView () and onViewCreated. These methods are performed when the fragment is run. As mentioned earlier, the view ID helps define this view separately from other views. Using findViewByID, your code can find random_button its ID, R.id.random_button. 2. Take a look atViewCreated. It customizes the clicker to random_button, which was originally created as the Next button. view.findViewById'lt;button'gt; (R.id.random_button).setOnClickListener - findNavController (R.id.action_FirstFragment_to_SecondFragment) - Here's what this code does: Use the findViewById method with the desired view ID as an argument, and then set the listener's click on that point of view. In the listener's body, use the action, which in this case to navigate another piece, and go there. (You'll find out later.) Just below that click the listener, add a code to set up the listener button for toast_button which creates and displays the toast. Here's the code: / find toast_button on his ID and set click listener view.findViewById (R.id.toast_button).setOnClickListener / create a toast with some text to appear in a short time qlt;val myToast and Toast.makeText (context, Hello Toast!, Toast.LENGTH_SHORT) Do you see a toasty message at the bottom of the screen? If you want, extract the message line into the resource, as it did for the button tags. You've learned that for an interactive view, you need to set up a clicker to view, and it says what to do when you press the view (button). Click listener can either: Implement a small amount of code directly. Call the method that determines the desired behavior of clicks in action. The method that shows toast is very simple; it does not interact with any other views in the layout. The next step is to add behavior to the layout to find and update other views. Updating the Graph button so that when pressed, the number on the screen increases by 1. In the layout file fragment_first.xml, look out for the textView ID: zlt;TextView android:id'id/textview_first in firstfragment.kt, add q click' listener' listener for the count_button's below the Other click' listeners' in's onviewcreated (). - countMe (view) - В классе FirstFragment добавьте количество методовMe (), которое принимает один аргумент View.</Button> </Button> </Button> </Button> The method is called at the touch of the Graph button and the clicker is called. Private Funny CountMe (View: View) - Use findViewById () to get TextView that shows the number. This method returns the view, so you have to drop the result in TextView. Provide a view ID to get textview_first. .../ Get a text view val showCountTextView - view.findViewById (R.id.textview_first) Get the value of showCountTextView. Get the text view value. val countString - showCountTextView.text.toString () Convert the value to the number, and increment it. .../ Converting the value to a number and incrementing its var count - countString.toInt () count to display a new value in TextView by software setting text property TextView. ... / Displaying a new value in text. showCountTextView.text - count.toString () Here's the whole method: private fun countMe (view: view) / Get a text view val showCountTextView - view.findViewById (R.id.textview_first) / Get the value of the text view. val countString - showCountTextView. text.toString () / Conversion of the value into a number and the increment of its var count - countString.toInt () count - count ... Displaying a new value in text-lt'lt'gt;representation. showCountTextView.text - count.toString Click the graph button and watch the account update. So far, you've focused on the first screen of the app. Next, you'll update the Random button to display a random number between 0 and the current account on the second screen. What you learn is how to pass the information on to the second snippet. The new snippet will show a headline and a random number on the screen. Here's what the screen will look like in the design view: %d indicates that part of the line will be replaced by a number. R is just a placeholder. Step 1: Add TextView for the occasional Open number fragment_second.xml (app zgt; res qgt; mock-up fragment_second.xml) and switch to the design view if necessary. Note that it has a ConstraintLayout that contains TextView and a button. Remove the chain limit between TextView and the button. Add another TextView from the palette and drop it in the middle of the screen. This TextView will be used to show a random number between 0 and the current number from the first snippet. Set the ID on q id/textview_random (textview_random the attribute bar.) Commite the top edge of the new TextView at the bottom of the first TextView, the left edge to the left of the screen, and the right edge to the right of the screen, and the bottom at the top of the previous button. Set both the width and height of the wrap_content. Install textColor, @android: color/white, set textSize to 72sp, and textStyle to Bold. Set the text to R. This text is only a placeholder until a random number is generated. Install 0.45. This TextView is limited at all edges, so it's better to use vertical slope than //TextView/textview/lt;textView/lt;textView/textView Position to help the layout look good on different screen sizes and orientation. Here's the XML code for TextView, which displays a random number: zlt;TextView android:id'id/textview_random android:layout_width'wrap_content android:layout_height'wrap_content android:text'R android:textcolor'@android:color/white android:7 The android:textstyle-bold app:layout_constraintbottom_totopof'id/button_second app:layout_constraintend_toendof'parent app:layout_constraintstart_tostartof'parent app:layout_constrainttop_tobottomof'id/textview_second app:layout_constraintvertical_bias'0.45/gt;lt;/TextView'gt; In fragment_second.xml, Select textview_second, which currently has the text Hello second snippet. Arg: %1$s per hello_second_fragment resource line. If android: the text is not configured, set it on the hello_second_fragment line. android:text @string/hello_second_fragment identifier change to textview_header in the attribute bar. Set the width match_constraint, but set the height to wrap_content, so that the height will change as needed to match the height of the contents. Place the upper, left and right fields to 24dp. The left and right borders can also be called start and end to support the localization of right and left languages. Remove any lower restriction. Set the text color to @color/colorPrimaryDark and text size to 24sp. In strings.xml, hello_second_fragment on Here's a random number between 0 and %d. Use Refactor zgt; rename... change the name of the hello_second_fragment random_heading. Here's the XML code for TextView that displays the headline: Step 3: Change the background color of the layout Give your new activity qlt;TextView android:id'id/textview_header android:layout_width'0dp android:layout_height'wrap_content android:layout_marginstart'24 androiddp:layout_marginleft'24dp'24dp'24dp'24dp'24 layout_margintop Dp android:layout_marginend/24dp android:layout_marginright/24dp android:text'@string/random_heading android:textcolor'@color/colorPrimaryDark android:textsize'24sp app:layout_constraintend_toendof'parent app:layout_constraintstart_tostartof'parent app:layout_constrainttop_totopof'parent'gt;'lt;'text;text A different background color, than the first action: In colors.xml, add a new color name to the screenBackground2'gt;resource: #26C6DA V/color Layout for the second activity, fragment_second.xml, set the background Of ConstraintLayout to a new color. In the attribute bar: Or in XML: android: background @color/screenBackground2 Your app now has a completed layout for the second snippet. But if you run the app and click random, it can cut out. The click handler set up for this Android Studio button needs some changes. In the next task, you'll study and fix the bug. When you were creating your project, you chose Basic Activity as the template for the new project. When Android Studio uses the Basic Activity template for a new he sets up two two and a navigation schedule to connect the two. It also sets the button to go from the first piece to the second. This is the button you changed to the Random button. And now you want to send a number when the button is pressed. Open nav_graph.xml (app zgt; res zgt; navigation nav_graph.xml). A screen that looks like a Layout editor in the Design view appears. It shows two fragments with arrows between them. You can zoom in with the buttons in the bottom right version, as it did with the Layout editor. You can move items freely in the navigation schedule. For example, if the snippets appear with SecondFragment on the left, drag FirstFragment to the left of SecondFragment so they appear in the order you're working with. Step 5: Include SafeArgs This will allow SafeArgs in Android Studio. Open Gradle Scripts (Project: My First App) Find the dependency section in the buildscript section and add the following lines after other classpath entries: def nav_version and 2.3.0-alpha02 classpath androidx.navigation:navigation-safe-args-gradle plug-in: $nav-version of Open Gradle Scripts build.gradle (Module: app) Just below the other lines that start with applying the plug-in add a line to SafeArgs: apply the plugin: 'androidx.navigation.safeargs.kotlin' Android Studio should display a message about Gradle files changing. Click Sync Now on the right side. Moments later, Android Studio should post in the Sync tab that it was a success: select the build of the Make Project. This should restore everything so that Android Studio can find FirstFragmentDirections. Troubleshooting: If the synchronization wasn't successful, confirm that you've added the right lines to the correct Gradle file. If there's still a problem, check out the developer's guide to Safe Args to update nav_version or other changes. On the navigation chart, click on FirstFragment and look at the Attribute panel on the right. (If the panel doesn't show, click on the vertical Attribute tag on the right.) The Action section shows what action will take place for navigation, namely switching to SecondFragment. Click on SecondFragment and look at the Attribute panel. You can't see anything in the Arguments section. Click on The Arguments of the Week. In add Argument, enter myArg for the name and set the type on Integer, then click Add. Step 6: Send the account to the second snippet of next/random button was created by Android Studio to go from the first snippet to the second, but it does not send any information. At this point you will change it to send the number for the current count. You'll get the current amount from the text that displays it, and pass that on to the second snippet. Open FirstFragment.kt (java app, zgt; com.example.myfirstapp. clicks to go from the first snippet to the second. Replace the code in this listener click with a line to find count the presentation of the text, textview_first. val showCountTextView - view.findViewById'lt;TextView(R.id.textview_first) Get the view text and convert it to Int. val currentCount - showCountTextView.text.toString (.toInt)) Create action with currentcount as an argument to actionFirstFragmentToSecondFragment (). val action - FirstFragmentDirections.actionFirstFragmentToSecondFragment (currentCount) Add a string to find a navigation controller and go with the action you've created. including the code you've added previously: override onViewCreated pleasure (view: View, Saved: Kit?) - super.onViewCreated (view, savedInstanceState) view.findViewById'lt;Button(R.id.random_button).setOnClickListener - val showCountTextView - view.findViewById'lt;TextView R.id.textview_first) val currentCount - showCountTextView.text.toString (.).).. navigation (action) / find toast_button by your ID view.findViewById (R.id.toast_button).setOnClickListener / create a toast with some text, to appear in a short time, i.e. Button and Toast.makeText, Hello Toast!, Toast.LENGTH_SHORT) / Show Toast myToast.show () - view.findViewById'lt;
Button(R.id.count_button).setOnClickListener - countMe (view) - Run your app. Click the Graf button several times. Now that you press the Random button, the second screen shows the correct line in the title, but still no account or random number, because you need to write code to do so. You wrote the code to send the current countdown to the second snippet. The next step is to add code to SecondFragment.kt to receive and use your current account. In SecondFragment.kt, add import for navArgs to the list of imported libraries. import androidx.navigation.fragment.navArgs In SecondFragment.kt before onViewCreated, add a line to determine where the arguments are. val args: SecondFragmentArgs by navArgs () In SecondFragment.kt below, where clicks listener is created, add lines to get the counting argument, get a line and format it with counting, and then set it to textview_header. val count - args.myArg val countText - getString (R.string.random_heading, count) view.findViewById<TextView>(R.id.textview_header). val random - java.util.Random () var randomNumber No 0 if (number of zgt; 0) - randomNumber - random.nextInt (account Number 1) - Add the code to convert that number into a string and install it as text for textview_random. view.findViewById<TextView>(R.id.textview_random). val args: SecondFragmentArgs by navArgs () redefine pleasure onViewCreated (view: View, savedInstoen: Bundle?) - super.onViewCreated (view savedInstanceState) </TextView> </TextView> </Button> </Button> </TextView> </Button> </TextView> </TextView> - val count - args.myArg val countText - getString (R.string.random_heading, (R.id.textview_header).text - countText val random - java.util.Random () var randomNumber No 0 if R.id.textview_random (counting zgt; 0) - randomNumber - random.nextInt (account No. 1) - viewViewById'lt Does the app show a random number in the new activity? Congratulations, you created your first Android app! The intention of this codelab was to get you to start building android apps. We hope you want to know a lot more though, for example, how can I save data? How do I do background tasks? How do I display a list of photos? How can I ... We encourage you to continue to study. We have more Android in Kotlin courses built by Google to help you on your way to learning Written tutorials Online courses Kotlin Bootcamp for programmers: This is an introduction to Kotlin for programmers. To do this, you need to gain some experience with object-oriented programming (Java, C, Python). These interactive, video courses were created by Google experts in collaboration with Udacity. Prepare these courses at your own pace in due course. It's time. </TextView></TextView>