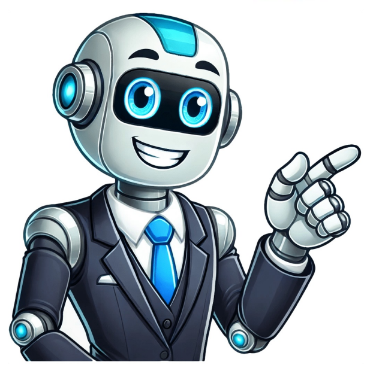

google-apps-script google-sheets Last updated: February 09, 2025 Google Apps Script enables you to extend Google Sheets and automate repetitive tasks. Once you write a script, you'll need to decide how to run it. There are several ways to do this so, in this tutorial, I will walk you through a few options and also help you understand when to use each one.Run scripts from the Apps Script editorThis is the simplest way to run a script. First, select the function you want to run from the dropdown menu and then click the Run button to run it. This method is most useful for testing scripts as you write them to check if they work correctly or for running one-off scripts that are used infrequently.The primary disadvantage of this method is that you need to open the script in the editor before running it and this isn't very user friendly. As you'll see below, there are many user friendly options for running scripts directly from the spreadsheet itself. Your browser does not support HTML5 video. Here is a link to the video instead. Run a script when the user selects a menu item from a custom menuYou can also run a function in your script whenever the user selects a menu item in a custom menu. A custom menu lets you extend the user interface of Google Sheets to make it easy for users to run scripts. Each menu item is linked to a function in your script and this function will be run whenever that menu item is selected by the user.This is a common and user friendly way to enable users to run scripts in Google Sheets. Consider using this approach for scripts that need to be explicitly run by the user. If on the other hand, you want to automatically run scripts, use triggers instead. Your browser does not support HTML5 video. Here is a link to the video instead. Run scripts from a custom sidebarYou can also run scripts when users interact with a custom sidebar in Google Sheets. A sidebar is another way to create a custom user interface in Google Sheets. A sidebar is useful if your script has several options that the user must configure in an interface OR if you need a custom widget that is persistently displayed within the Google Sheets UI.You can run a function in your script from the sidebar by using the google.script.run API. Your browser does not support HTML5 video. Here is a link to the video instead. Run scripts from a modal dialogA modal dialog is used to display a custom user interface built using HTML as an overlay. Unlike a sidebar, a modal dialog prevents the user from interacting with the spreadsheet. Therefore, it should only be used when the user must see some contextual information or provide contextual input. You can use the google.script.run API to run a function in your script from a modal dialog. Your browser does not support HTML5 video. Here is a link to the video instead. Run Apps Script with a single click by using an image or a drawing in your spreadsheetYou can also create clickable buttons within your spreadsheet. This is especially useful when you are creating simple applications within a Google Sheets spreadsheet that require the user to run a script to take some action and you want to ensure they do not miss this step.Unlike sidebars and custom menus, you can place these call-to-action buttons within your spreadsheet itself and you can also style them to ensure they stand out. In the video below, I'm sure you will not miss seeing the green and blue buttons. However unless you look closely, you might miss seeing the custom "Settings" menu.You can create a clickable button by inserting an image or a drawing in your spreadsheet and assigning a script to it. This will enable users to run the script with a single click. Simple and efficient! Your browser does not support HTML5 video. Here is a link to the video instead. Run the script periodically or whenever some event occurs by using triggersThe previous examples in this tutorial all required the user to take some action to run the script. There are many instances where you might want scripts to be run automatically. For example, you might want to run your script whenever the spreadsheet is edited or whenever a form is submitted. You can achieve this by using triggers. A trig works by running a function that you specify whenever an event occurs or periodically based on a schedule that you specify.The video below shows an example of a trigger in action. Here, a toast notification is automatically displayed whenever the spreadsheet is edited. Your browser does not support HTML5 video. Here is a link to the video instead. Run your script as a custom function from a formula in your spreadsheetAnother common use case for scripting in Google Sheets is writing your own functions that can be used within formulas in your spreadsheet. If you've used spreadsheets, you've probably used functions like SUM(), AVERAGE(), etc., in formulas. You can write your own custom functions using Apps Script and then run them from a formula.In the video below, I'm using a custom function called HELLO() that simply returns the string "Hello!". Your browser does not support HTML5 video. Here is a link to the video instead. Create a script by recording a macro and run it from the menu barA macro is a feature in Google Sheets where you can create a script by recording actions such as formatting a range. You can then run the macro to apply the same set of actions in the future to a different range. You can run a previously recorded macro by selecting it from the Tools —> Macros menu. Your browser does not support HTML5 video. Here is a link to the video instead. Run the script from an Apps Script web applicationGoogle Apps Script can also be used to create simple web applications! You can use this feature to create custom dashboards or even internal applications. You can run scripts from these web applications by using the google.script.run API.In the video below, I've created a simple web application that has a single button that you can click to reset the spreadsheet. The button click runs a function that resets the spreadsheet. Your browser does not support HTML5 video. Here is a link to the video instead. ConclusionIn this tutorial, I walked you through several ways to run scripts in Google Sheets.Run one-off scripts from the Apps Script code editor: Use this method to test scripts as you write them or to run one-off scripts on an ad-hoc basis.Run the script whenever users select a menu item on a custom menu: This is a common and user friendly way to enable users of your spreadsheet to run custom scripts.Run scripts from a custom sidebar by using the google.script.run API: Use this method to add interactivity to your custom sidebar.Run scripts from a modal dialog by using the google.script.run API: Use this method to add custom logic or other interactive functionality to a modal dialog.Run a script whenever users click a drawing or an image in your spreadsheet: Use this method to create Call-To-Action (CTA) buttons in your spreadsheet.Run scripts periodically or whenever a specific event occurs using Triggers: Use triggers to automatically run scripts without any user intervention.Run a custom function from a formula in your spreadsheet: Create your own functions using Apps Script that users can run from a formula in the spreadsheet.Create and run macros in your spreadsheet: Create a script by recording a macro and run it from the Tools —> Macros menu.Run scripts from an Apps Script web application by using the google.script.run API: Use this method to add interactive functionality to an Apps Script web application. Your feedback helps me create better content DISCLAIMER: This content is provided for educational purposes only. All code, templates, and information should be thoroughly reviewed and tested before use. Use at your own risk. Full Terms of Service apply. Join 1,500+ professionals who are supercharging their productivity with Google Sheets automation Exclusive Google Sheets automation tutorials and hands-on exercises Ready-to-use scripts and templates that transform hours of manual work into seconds Email updates with new automation tips and time-saving workflows By subscribing, you agree to our Privacy Policy and Terms of Service Have you ever found yourself working in Google Sheets, wishing you could automate a repetitive task or customize a function to suit your unique needs? If so, you're not alone. Many users discover that Google Sheets' built-in functionalities, while comprehensive, don't always fit their specific requirements. That's where Google Apps Script comes into play. In this article, we'll walk through how to enable and use Apps Script in Google Sheets, giving you the power to automate complex tasks, create custom menus, and much more. Whether you're a beginner or looking to expand your skills, this guide will provide the insights you need to get started.Build dashboards & reports in seconds with the best AI spreadsheet. Bricks makes creating dashboards, reports, and charts a breeze. Try it free → Before we jump into the practical steps, let's take a moment to understand what Google Apps Script is all about. Think of it as the secret sauce that allows you to enhance your Google Sheets experience. Essentially, Google Apps Script is a cloud-based scripting language based on JavaScript. It's designed to extend the capabilities of all Google Workspace products, including Sheets, Docs, and Forms. What makes Apps Script so appealing? Well, it's versatile and user-friendly. You don't need a background in coding to start tinkering with it. With just a few lines of script, you can automate repetitive tasks, create custom functions, and even build simple applications. Imagine creating a custom menu to sort your data with a single click or setting up an automated email notification whenever a specific cell value changes. The possibilities are endless, and Apps Script makes it all possible. Now that we've covered the basics, it's time to get our hands dirty. Accessing Google Apps Script is surprisingly straightforward. Here's how you can get started: Open a Google Sheet document where you want to use Apps Script. Click on Extensions in the top menu bar. Select Apps Script from the dropdown menu. This action will open a new tab in your browser, displaying the Apps Script editor. And just like that, you're in! The editor might look a bit intimidating at first, but don't worry. It's simpler than it appears. The left pane lists your project files, while the main area is where you'll write and edit your scripts. You'll also notice a toolbar with options to run, debug, and save your scripts.Bricks is an AI-first spreadsheet that makes creating dashboards, reports, and analyzing your data a breeze. Import your data, describe what you want, and improve it by asking for changes.GET STARTED FOR FREE Getting started with your first script is a lot like riding a bike—once you get the hang of it, you'll wonder why you ever found it challenging. Let's create a simple script to see how it works: In the Apps Script editor, delete any pre-existing code in the main script file, usually named Code.gs. Copy and paste the following code into the editor: function myFirstScript() { Logger.log('Hello, Google Apps Script!'); } Click the disk icon to save your work. To run the script, click on the play button (run icon). You might be prompted to authorize the script, so go ahead and do that. After running the script, open the Logs under the View menu to see the output. You should see your "Hello, Google Apps Script!" message there, confirming that your script ran successfully. Now that you have a basic understanding of writing a script, let's talk about triggers. In the context of Apps Script, triggers are a way to automate script execution based on specific events. For instance, you might want to run a script whenever a form is submitted or a spreadsheet is edited. Google Apps Script provides two types of triggers: Simple Triggers: These are easy to set up and don't require user authorization. Examples include onOpen and onEdit. Installable Triggers: These offer more flexibility and can respond to a wider range of events. However, they do require user authorization. To add a trigger, click on the clock icon in the Apps Script editor. This will open the triggers dashboard, where you can create and manage your triggers. Simply follow the prompts to set up the conditions under which your script should run. Automation is where Apps Script truly shines. Consider this common scenario: You have a Google Sheet that tracks project tasks, and you want to automatically send an email alert when a task is marked as complete. Here's a simple way to achieve that: Write a script that checks the status of tasks in your sheet. Use the MailApp.sendEmail function to send an email notification whenever a task is marked as "Complete." function sendEmailAlert() { var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(); var range = sheet.getRange('A2:A10'); // Range containing task statuses var values = range.getValues(); for (var i = 0; i < values.length; i++) { if (values[i][0] === 'Complete') { MailApp.sendEmail('youremail@example.com', 'Task Completed', 'A task has been completed!'); } } } By setting up a trigger to run this script daily or whenever the sheet is edited, you can ensure that you're always informed of task completions without lifting a finger. Have you ever wished that Google Sheets had a specific function to perform a calculation or task unique to your needs? With Apps Script, creating custom functions is a breeze. Let's say you want a function that calculates the square of a number. Here's how you can do it: In the Apps Script editor, create a new function: function calculateSquare(number) { return number * number; } Save your script. Go back to your Google Sheet, and use your new custom function just like any other built-in function: =calculateSquare(5) Custom functions operate just like native functions in Google Sheets, allowing you to extend the application's capabilities to meet your specific requirements. Google Sheets is great, but wouldn't it be even better if you could customize the interface to include your own handy features? That's precisely what custom menus allow you to do. By adding a custom menu to your Google Sheets, you can make your scripts more accessible and user-friendly. Here's a quick example of how to create a custom menu: In the Apps Script editor, modify your script to include a function that adds a menu: function onOpen() { var ui = SpreadsheetApp.getUi(); ui.createMenu('My Custom Menu') .addItem('Say Hello', 'myFirstScript') .addToUi(); } Save your script and reload your Google Sheet. You'll notice a new menu item called "My Custom Menu" in the top menu bar. Click on it to run your script. Custom menus are a simple yet powerful way to make your scripts more interactive and convenient for users, especially if you're sharing the sheet with others. No matter how experienced you are, running into errors while scripting is almost inevitable. The good news is that Apps Script provides tools to help you debug and test your scripts efficiently. Here are some tips for debugging your Apps Script code: Log Statements: Use the Logger.log function to print variable values and messages to help track down issues. Breakpoints: Insert breakpoints in your code to pause execution and inspect the current state of your script. Execution Transcript: View the execution transcript to see a step-by-step account of what your script did and where it might have gone wrong. By systematically testing your scripts and identifying errors, you can ensure that your automated processes run smoothly and without a hitch. Once you've got a script you love, you might want to share it with others or collaborate on it as part of a team project. Fortunately, Google Apps Script makes this easy. You can share your script projects with others just like you would a Google Doc or Sheet. Click the Share button in the Apps Script editor. Add collaborators by entering their email addresses. Set permissions to control whether they can view or edit the script. Collaborative editing is a powerful feature that allows multiple users to work on the same script in real-time, making team projects more efficient and productive.Bricks makes it easy to analyze data, create dashboards and reports, and get insights from your spreadsheet data.SIGN UP for free If you're feeling adventurous and want to push the limits of what you can do with Apps Script, there are several advanced features worth exploring. For instance, you can: Connect to External APIs: Use the UrlFetchApp service to interact with third-party APIs and gather data from external sources. Build Web Apps: Create web applications that run on Google Cloud, accessing Google Sheets as a backend database. Integrate with Other Google Services: Automate tasks across different Google Workspace applications, such as sending emails through Gmail or creating calendar events in Google Calendar. These advanced features open up a world of possibilities, allowing you to build powerful applications and automate complex workflows. We've covered a lot of ground here, from the basics of accessing Google Apps Script to creating custom functions and debugging your code. Hopefully, you now feel more confident in using Apps Script to enhance your Google Sheets experience and automate tasks that once felt tedious. For those looking to take their productivity to the next level, Bricks offers an exciting opportunity. With its AI-powered capabilities, Bricks integrates spreadsheets, docs, and presentations into one seamless tool. Whether it's writing formulas, cleaning data, or creating charts, Bricks AI can handle it all, saving you time and effort. Plus, it can generate visuals like dashboards and project trackers, all linked with your documents and presentations. It's a tool that makes working with data not just easier but more intuitive and connected.header-1-2header-2-2header-1-4header-2-4header-3-4header-4-4header-5-4header-1-6header-2-6header-1-8header-2-8Bricks is the AI-first dashboard and reporting tool for spreadsheet data. Create charts, graphs, and analyze your data in seconds - no data analyst needed.Create your first report Last Updated on October 5, 2022 by Jake Sheridan In this tutorial, you will learn how to run a script in Google Sheets. Google Sheets is a powerful tool right out of the box, but it becomes even more powerful once you begin using scripts to add additional, custom features to your spreadsheets. Scripts can be used to process data, apply formatting, and much more. This tutorial outlines how to run an already-written script, but does not go into details of developing scripts Script SetupScript SetupRunning Your ScriptRun a Script from a ButtonSummary Before running a script, you have to add it to your Google Sheet Step 1 Open the Extensions Menu and choose Apps Script to launch the Script Editor. If you haven't switched to the new menu layout, you'll find the Script Editor in the Tools menu. Step 2 The Script Editor will load in a new browser tab. Delete the existing text from the editor region Step 3 Type in or copy paste the script you want to run. If you don't have a script already, you can use the following code that writes "Hello World" to the selected cell. function helloWorld() { const cell = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet().getActiveRange(); cell.setValue("Hello World"); } Step 4 Add this tag to your code to make sure the script only affects the current spreadsheet /** * @OnlyCurrentDoc */ Step 5 Name your script project by clicking on the heading that says Untitled Project then typing in a descriptive name. For this example, we'll call the project "Hello World". Step 6 Click the save button to save your script. Step 7 The script is now ready to run Running Your Script The first time you run a script in Google Sheets under a given Google Account, you'll have to authorize the code to run Here's how: Step 1 In the script editor, make sure the script you want to run is selected in the Select Function to Run dropdown, then click the Run button Step 2 Wait for the popup to appear, then click Review Permissions Step 3 Step 4 Click on the account you want to run the script under Step 5 Review the listed permissions, then click Allow if you are ok with granting the script these permissions and trust the source Step 6 The script is now authorized. Click Run one more time to run the script and paste "Hello World" into the active cell(s) Run a Script from a Button You can run scripts from the script editor at any time, but it's easier to run them directly from the spreadsheet. One way to do this is by creating a button in Google Sheets. Here's how: Step 1 Open the Insert menu and choose Drawing Step 2 Use the Drawing Editor tools to create a button, then click Save and Close to add it to your spreadsheet Step 3 Select the button and click on the 3 dots that appear to open the menu, then choose Assign Script Step 4 Type the name of the function you want to run exactly as it appears in the Script Editor. Then click Ok. Step 5 You can now run the script by clicking the button Summary Example Spreadsheet: Make a copy of the example spreadsheet In this tutorial, I covered how to run a script in Google Sheets. Want more? Check out all the Google Sheets Tutorials. google-apps-script google-sheets Last updated: February 09, 2025 Google Apps Script enables you to extend Google Sheets and automate repetitive tasks. Once you write a script, you'll need to decide how to run it. There are several ways to do this so, in this tutorial, I will walk you through a few options and also help you understand when to use each one.Run scripts from the Apps Script editorThis is the simplest way to run a script. First, select the function you want to run from the dropdown menu and then click the Run button to run it. This method is most useful for testing scripts as you write them to check if they work correctly or for running one-off scripts that are used infrequently.The primary disadvantage of this method is that you need to open the script in the editor before running it and this isn't very user friendly. As you'll see below, there are many user friendly options for running scripts directly from the spreadsheet itself. Your browser does not support HTML5 video. Here is a link to the video instead. Run a script when the user selects a menu item from a custom menuYou can also run a function in your script whenever the user selects a menu item in a custom menu. A custom menu lets you extend the user interface of Google Sheets to make it easy for users to run scripts. Each menu item is linked to a function in your script and this function will be run whenever that menu item is selected by the user.This is a common and user friendly way to enable users to run scripts in Google Sheets. Consider using this approach for scripts that need to be explicitly run by the user. If on the other hand, you want to automatically run scripts, use triggers instead. Your browser does not support HTML5 video. Here is a link to the video instead. Run scripts from a custom sidebarYou can also run scripts when users interact with a custom sidebar in Google Sheets. A sidebar is another way to create a custom user interface in Google Sheets. A sidebar is useful if your script has several options that the user must configure in an interface OR if you need a custom widget that is persistently displayed within the Google Sheets UI.You can run a function in your script from the sidebar by using the google.script.run API. Your browser does not support HTML5 video. Here is a link to the video instead. Run scripts from a modal dialogA modal dialog is used to display a custom user interface built using HTML as an overlay. Unlike a sidebar, a modal dialog prevents the user from interacting with the spreadsheet. Therefore, it should only be used when the user must see some contextual information or provide contextual input. You can use the google.script.run API to run a function in your script from a modal dialog. Your browser does not support HTML5 video. Here is a link to the video instead. Run Apps Script with a single click by using an image or a drawing in your spreadsheetYou can also create clickable buttons within your spreadsheet. This is especially useful when you are creating simple applications within a Google Sheets spreadsheet that require the user to run a script to take some action and you want to ensure they do not miss this step.Unlike sidebars and custom menus, you can place these call-to-action buttons within your spreadsheet itself and you can also style them to ensure they stand out. In the video below, I'm sure you will not miss seeing the green and blue buttons. However unless you look closely, you might miss seeing the custom "Settings" menu.You can create a clickable button by inserting an image or a drawing in your spreadsheet and assigning a script to it. This will enable users to run the script with a single click. Simple and efficient! Your browser does not support HTML5 video. Here is a link to the video instead. Run the script periodically or whenever some event occurs by using triggersThe previous examples in this tutorial all required the user to take some action to run the script. There are many instances where you might want scripts to be run automatically. For example, you might want to run your script whenever the spreadsheet is edited or whenever a form is submitted. You can achieve this by using triggers. A trig works by running a function that you specify whenever an event occurs or periodically based on a schedule that you specify.The video below shows an example of a trigger in action. Here, a toast notification is automatically displayed whenever the spreadsheet is edited. Your browser does not support HTML5 video. Here is a link to the video instead. Run your script as a custom function from a formula in your spreadsheetAnother common use case for scripting in Google Sheets is writing your own functions that can be used within formulas in your spreadsheet. If you've used spreadsheets, you've probably used functions like SUM(), AVERAGE(), etc., in formulas. You can write your own custom functions using Apps Script and then run them from a formula.In the video below, I'm using a custom function called HELLO() that simply returns the string "Hello!". Your browser does not support HTML5 video. Here is a link to the video instead. Create a script by recording a macro and run it from the menu barA macro is a feature in Google Sheets where you can create a script by recording actions such as formatting a range. You can then run the macro to apply the same set of actions in the future to a different range. You can run a previously recorded macro by selecting it from the Tools —> Macros menu. Your browser does not support HTML5 video. Here is a link to the video instead. Run the script from an Apps Script web applicationGoogle Apps Script can also be used to create simple web applications! You can use this feature to create custom dashboards or even internal applications. You can run scripts from these web applications by using the google.script.run API.In the video below, I've created a simple web application that has a single button that you can click to reset the spreadsheet. The button click runs a function that resets the spreadsheet. Your browser does not support HTML5 video. Here is a link to the video instead. ConclusionIn this tutorial, I walked you through several ways to run scripts in Google Sheets.Run one-off scripts from the Apps Script code editor: Use this method to test scripts as you write them or to run one-off scripts on an ad-hoc basis.Run the script whenever users select a menu item on a custom menu: This is a common and user friendly way to enable users of your spreadsheet to run custom scripts.Run scripts from a custom sidebar by using the google.script.run API: Use this method to add interactivity to your custom sidebar.Run scripts from a modal dialog by using the google.script.run API: Use this method to add custom logic or other interactive functionality to a modal dialog.Run a script whenever users click a drawing or an image in your spreadsheet: Use this method to create Call-To-Action (CTA) buttons in your spreadsheet.Run scripts periodically or whenever a specific event occurs using Triggers: Use triggers to automatically run scripts without any user intervention.Run a custom function from a formula in your spreadsheet: Create your own functions using Apps Script that users can run from a formula in the spreadsheet.Create and run macros in your spreadsheet: Create a script by recording a macro and run it from the Tools —> Macros menu.Run scripts from an Apps Script web application by using the google.script.run API: Use this method to add interactive functionality to an Apps Script web application. Your feedback helps me create better content DISCLAIMER: This content is provided for educational purposes only. All code, templates, and information should be thoroughly reviewed and tested before use. Use at your own risk. Full Terms of Service apply. Join 1,500+ professionals who are supercharging their productivity with Google Sheets automation Exclusive Google Sheets automation tutorials and hands-on exercises Ready-to-use scripts and templates that transform hours of manual work into seconds Email updates with new automation tips and time-saving workflows By subscribing, you agree to our Privacy Policy and Terms of Service There are a lot of great things you can do with in-built functions and functionalities in Google Sheets. But there are a few things that are either not possible by default or require you to take a series of steps to achieve it. With Google Sheets Scripts, you can automate a lot of stuff and even create new Google Sheets script functions that you wish existed. In this article, I will cover the basics of Google Apps Script with some simple yet practical examples of using scripts in Google Sheets. What is Google Apps Script (GAS)? Google Apps Script is a coding language and allows you to create automation and functions for Google Apps (which can include Google Sheets, Google Docs, Google Forms, Drive, Maps, Calendar, etc.) In this tutorial (and on this site), I will be focussing on using Script for Google Sheets. However, most of what I cover here can also be used for other Google Apps. This Google Apps Script (GAS) coding language uses Javascript and is written in the back-end of these Google Sheets (there is a neat interface that allows you to write or copy/paste the code in the back-end). Since Google Sheets (and other Google Apps) are cloud-based (i.e., can be accessed from anywhere), your Google Apps Script is also cloud-based. This means that if you create a code for a Google Sheets document and save it, you can access it from anywhere. It doesn't reside on your laptop/system but on Google cloud servers. What Makes Google Apps Script Useful? There are many good reasons you may want to use Google Apps Scripts in Google Sheets. We'll cover a few use cases in our Google Script tutorial, and there are plenty more such as how it: Allows You to Automate Stuff Suppose you regularly download data from any tool or database and have to combine and clean it in Google Sheets. Usually, data cleaning and combining it involves a series of steps. This may not be a big deal if you have to do it only a few times, but if you have to do it quite often, automating these steps can save you a lot of time. And that's what you can do with Google Apps Script. All you need to do is invest some time in getting the code ready once, and whenever you have to do the same steps again, you simply run the script code in Google Sheets and let GAS do all the heavy lifting for you. Prefer a no-code option? Free extensions like Coefficient sync live data from your business systems into Google Sheets. Easily sync and blend data from CRMs, BI tools, databases, and payment platforms right inside your spreadsheet. Think systems like Salesforce, Hubspot, Google Analytics, Tableau, Looker, MySQL, Stripe, and Shopify. Can You Create New Functions in Google Script Language? There are already hundreds of awesome functions in Google Sheets, and in most cases, these function should be enough. But in some cases, you may want some additional functionality which can not be achieved with the inbuilt function (or it can be but the formula become huge and complicated). In such cases, you can quickly write a GAS code to create a custom function. These functions can be used just like regular functions in the Google Sheets document and instantly makes your work easier. Can Talk To Other Google Apps Since Google Apps Script is the common coding language behind a lot of Google Apps, you can use it to interact with other apps as well. For example, if you have 10 Google Sheets documents in your Google Drive, you can use GAs to combine all these and then delete all these Google Sheets documents. This is possible because you can use GAS to work with multiple Google Apps. Another useful example of this could be to use data in Google Sheets to quickly schedule reminders in your Google Calendar. Since both of these apps use GAS, this is possible. Extend the Functionality of Google Sheets Apart from automating stuff and creating functions, you can also use the GAS to enhance the functionality of Google Sheets. While Google Sheets provides a lot of functionality for everyone, you can also code something that's more suited to your needs. And since you can reuse the code, again and again, it makes you a lot more productive and efficient. Getting Started with Google Sheets Script Editor The script editor in Google Sheets is the place where you can write scripts and then run it. There would be a separate script editor for different Google Apps. For example, in the case of Google Forms, there would be a 'Script Editor' where you can write and execute code for Google forms. Where Is the Script Editor in Google Sheets? In Google Sheets, you can find the Script Editor in the Extensions tab. Anatomy of the Google Sheet Script Editor Once you click on the Script Editor option, it opens the Script Editor in a new window (as shown below). You can change the name of the Project by clicking on the top-left part of the screen that says 'Untitled project'. When you click on it, it will open a dialog box where you can enter the name of the project. It takes a few seconds to implement the name change. Within a script project, you can have multiple script file. For example, if you have three different things that you want to automate in Google Sheets and you create three different scripts for it, you can have all the three different scripts in the same Project file. In the left pane of the Project, you have the default script file – Code.gs. This script file is where you can write code. You can have multiple scripts in the same script file and you can have multiple script files as well. If you click on the small downward-facing arrow at the right of the Script file name, it will show you options to rename, delete, and create a copy of the script file. Note: You always need to have at least one script file in the project. If you only have one, it will not let you delete it. On the right side of the script file is the code window where you can write the code. Script Editor Toolbar The Script Editor Toolbar has the following options: Redo/Undo button: To redo/undo changes you have done in the script Indentation button: This is a toggle button and you can enable or disable indentation by clicking on it. When indentation is enabled, it automatically indents some parts of your script to make it more readable. This can be the case when you're using loops or IF statements. It will automatically indent sets of codes within the loop to increase the readability (if indentation is enabled). This option is enabled by default and I recommend you keep it that way Save button: You can use this button to save any changes to your script. You can also use the keyboard shortcut Control + S. Note that unlike Google Sheets, you need to save your project to make sure the changes are not lost. Current Project Trigger's button: When you click on this button, it will open the Triggers dashboard that lists all the triggers you have. A trigger is anything that triggers the running of code. For example, if you want a code to run and enter the current date and time in cell A1 whenever someone opens the Google Sheets, then you will use a trigger to do this. Run button: Use this to run the script. In case you have multiple functions, select any line in the one you want to run and then click on the Run button. Debug button: Debug helps you find errors in your code and also gives you some useful information. When you click on the Debug button, it also shows some additional debugging-related options in the toolbar. Select Function: This is a drop-down that would list all your functions in the script file. This is useful when you have many functions in the script and want to run a specific one. You can simply select the name from here and then click on the run button (or debug it if you want). Google Sheets Script Examples Let's take a look at a couple of real-life examples of how Google Sheets Script can help everyday users. Automatically Generate Charts With Google Spreadsheets Script For this example, we can pretend that you're working on similar sheets month to month that come with new data each time. With each new set of data, you want to create an easy-to-read chart without having to go into the charts menu every time. You can accomplish this with Google Sheets Script. In our imaginary scenario, we'll pretend you need to compare sales statistics for each employee every month. We'll use the following data set: We could use the following script to automatically create a graph with the data in each new sheet. function SalesChart() { var spreadsheet = SpreadsheetApp.getActiveSpreadsheet(); var sheet = spreadsheet.getSheets()
[0]; var SalesChart = sheet.newChart() .setChartType(Charts.ChartType.LINE) .addRange(sheet.getRange('A2:B10')) .setPosition(5, 5, 0, 0) .build(); sheet.insertChart(SalesChart); } All you have to do is: Navigate to the Google Sheets Script Editor Enter in the code, if you're using different cell references you can change them (highlighted in blue) or change the chart type (highlighted in green) Click Run (in the red box) Check back on your sheet to see if the script ran successfully It looks like ours ran with no issues, so happy days! Create a Custom Formula With Google Apps Script Google Sheets comes loaded with plenty of formulas right off the bat. But, what if you work with a formula constantly that is missing from their formula library? You can use a spreadsheet script to save you from having to recreate your formula every time. A simple example of this is converting Celsius to Fahrenheit. You can add this formula to your spreadsheet with the following steps. Navigate to the script editor as in the previous example Name the function, its best to name it the same thing you intend to type after the = sign in sheets Copy this function into your script editor (or create your own function) and then click run function CTOFH (input) { return input * 1.8 + 32; } You should now be able to use the function in your spreadsheet Note that the shortcut you created won't come up in the suggested ones, so make sure its memorable for your own sake. Google Script Editor Menu Options Apart from the toolbar, there are many other options available in Google Apps Script in Google Sheets. While the most used options are already a part of the toolbar, there are some more options in the menu that you may need when you start working with GAS. In this section of this article, I will cover each menu option and some of the options it has. You can play around with the options on your own to get a better idea. FILE: From the File menu, you can add a new project or a script file. A project would be a completely new project in a separate window where you can create more script files. When you add a new script file, it simply adds it to the same project (you'll be able to see in the left pane below your current script files). You can also rename and delete projects from here. Another useful option you can find in the File menu is the ability to manage versions of the projects. When you save a project, a version of it is saved and you can go back and revisit that version if you want. EDIT: Edit has some useful options that can help when you are writing or editing the code. For example, there is an option to find and replace text in your code. There are also options such as Word Completion, Content Assist, and Toggle comments. VIEW: This has options that can be useful when you want to get more information about the script when it has executed or want to add logs to help in debugging in the future. For example, you can get the execution transcript, which details all the actions done by your script. RUN: There are options to run different functions or debug them. Since these options are also available on the toolbar, it's less likely to be used from the menu PUBLISH: This has more advanced features such as publishing your scripts as web-apps. RESOURCES: This gives you access to advanced options such as Libraries and Advanced Google Services. You can use these options to connect with other Google properties such as Google Forms or Docs. HELP: This has tutorials and resources that can help you when you are getting started/working with Google Apps Scripts. One of the most useful options here is the link to the Documentation page where you can find a lot of guides and references to learn Google Apps Scripts In this article, I have covered the basics of Google Apps Script and the overall anatomy of the interface. We hope this Google Sheets Script tutorial was useful for you. You May Also Like the following tutorials: Apps Script is a rapid application development platform that gives you the ability to automate, customize, and extend your Google Workspace experience. With Apps Script, you can save time and effort by streamlining onerous or complex work within Google Workspace. Apps Script features include the following: Apps Script's built-in services let you read, update, and manipulate your Google Workspace application data with scripts. You can create scripts using Apps Script's in-browser code editor—there's no need to install or run code development software. You can design user interfaces for Google Workspace editors that let you activate scripts directly from those editors using menu items, dialogs, and sidebars. This Fundamentals of Apps Script with Google Sheets codelab playlist teaches the basics of Apps Script and how to use it to improve your Google Sheets experience. This codelab focuses on teaching Apps Script basics. The Spreadsheet service You can use Apps Script to extend Google Sheets to save time and effort. Apps Script provides the Spreadsheet service which allows scripts to interact with your Google Sheet files and the data they contain. You can use this service to automate the following common spreadsheet tasks: Create or modify a spreadsheet. Read and update cell data, formulas, and formatting. Create custom buttons and menus. Import and export data from other Google applications or third-party sources. Share and control access to spreadsheets. What you'll learn This playlist covers all the topics you'll need to get started using Apps Script with Google Sheets: The codelabs in this playlist are meant to be read in order, so start with this one and complete them in sequence for the best learning experience. Move to the next section to learn more about this codelab's contents. Welcome to the first codelab of this playlist. In this codelab, you'll learn the basics of using Apps Script with Google Sheets. Specifically, this codelab focuses on two key concepts: macros and custom functions. A macro is a series of recorded actions within Google Sheets. Once recorded, you can activate a macro to repeat those actions later with a menu item or shortcut key. You can both create and update your own macros in both Google Sheets and the Apps Script code editor. In the Apps Script code editor, you can also create custom functions. Similar to the built-in functions Sheets offers (such as SUM or AVERAGE), you can use Apps Script to write your own custom functions for simple and niche operations (such as conversions or string concatenation). Once created, you can call these functions in Sheets as you would a built-in function. Custom functions can also be used in cell formulas you write, combining them with other functions as needed. Read below to learn what concepts and requirements this codelab involves. What you'll learn How to create a script for Google Sheets. How to navigate the Apps Script editor. How to create and update macros. How to create your first Sheets custom function. What you'll need Note: Apps Script is based on JavaScript. If you're new to JavaScript, Codecademy offers a free Introductory JavaScript course. While Google didn't build this course, it should cover everything you need to know. You're done with the introductions. Go to the next section to start working with macros. Typically, when working in spreadsheets, you can enter into a loop of repetitive actions—copying cell values, formatting, creating formulas, and so forth—which can grow tedious and lead to mistakes. To automate repeated actions, Google Sheets provides macros. Macros let you 'record' a series of actions in a sheet. With a recorded macro, you can repeat the same actions elsewhere in a spreadsheet with a simple hotkey press. In this section, you'll learn how to build a macro in Sheets. In the next section, you'll see how macros are built using Apps Script. Before you begin Before you continue, you need a spreadsheet with some data. We've provided one for you: click this link to copy the data sheet and then click Make a copy. A copy of the example spreadsheet for you to use is placed in your Google Drive folder and named "Copy of Top 10 Highest Grossing Films (2018)." Create a macro Now that you have a spreadsheet to work with, you can record a macro in Google Sheets. For this example, you'll create a macro that formats a header row for your data. Just follow these steps: Click cell A1 to place the cursor in the row. This is your header row. In the menu, select Extensions > Macros > Record Macros. Once you start recording, Google Sheets remembers every action you take within the spreadsheet: highlighting cells, adding data, switching to different sheets, formatting, and so forth. These actions become the 'script' that gets repeated once you save and activate the macro later. In the Macro dialog, select Relative reference. Key Terms: Macros can use two types of references in Sheets to refer to specific cells. Absolute references are fixed to the exact locations recorded (for example, cell A5), while relative references are applied using the user's current selection as a starting point (for example, the fourth cell down from the current selection). This codelab asks you to use relative references for your macros. Recolor the top row's Fill Color from white to dark magenta 3. Change the top row's Text Color from black to white. To bold the text, press Ctrl+B (or Cmd+B on macOS). To freeze the top row, select View > Freeze > 1 row. Click Save in the macro dialog. A new dialog asks you to name the macro. Enter the name "Header" and click Save. Using Sheets' user interface, you've made a macro specialized for formatting headers. Activate your macro You can apply your new macro in Sheets by following these instructions: To create a sheet, click Add Sheet . In the new sheet, add some text to A1:C2. Feel free to follow the example inputs below: To apply the macro to the selected area, click Extensions> Macros> Header. Authorize the macro by following the on-screen instructions. Note: If you're using a gmail.com account, you might also get an unverified app dialog when you first use your script. Google uses this to warn users who may be using code from unknown or untrusted authors. If you see this dialog, it's OK to proceed since you're the script author. Follow the on-screen prompts to continue authorizing the script. The process of getting this permission is called authorization. Repeat Step 4 to run the macro again (authorizing it stops the first execution). Congrats, you've learned how to apply macros in Sheets. Your spreadsheet should look like this: Macros allow you to create spreadsheets efficiently, and, in the next part of this codelab, you'll learn how to make your macros even more powerful. Here's the secret: when you record a macro, what you're really doing is writing Apps Script code. Behind the scenes, Sheets constructs the code that matches the macro actions. In the next section, you'll learn how to modify the code directly using Apps Script's in-browser editor. As you create a macro, Google Sheets saves your actions as an Apps Script function. When you activate the macro, Google Sheets calls the Apps Script function to apply those actions in the same order. The script editor Now that you've created a macro, you can look at its code. To view the macro script, click Extensions > Apps Script to open the browser code editor for Apps Script. The script editor allows you to write code in Apps Script and to run those scripts on Google servers. Note: The macros and scripts you're creating in this codelab are attached to the Google Sheet file they operate on, and can be accessed at any time from the Sheets Extensions > Apps Script menu. Scripts attached to a Google Sheet like this are called container bound. Analysis of macros.gs Review the current script. Sheets created the macros.gs script file when you recorded the Header macro, filling it with a corresponding Apps Script function called Header. When you activate the Header macro, Sheets runs this function. Look at the image below to get familiar with the structure of your macro function in Apps Script. If you recorded the steps in a different order, or clicked around the spreadsheet while recording, your code might look a little different than this. The first line is an annotation comment that affects authorization: /** @OnlyCurrentDoc */ Most scripts ask the user for some permissions before they can run. These permissions control what the user is allowing the script to do. When the @OnlyCurrentDoc comment is present in a script project, Apps Script only asks for permission to access and update the current spreadsheet. Without this comment, Apps Script would ask permission to access and update all of the user's spreadsheets. It's always best practice to include this annotation when you're only working with a single file. The macro recorder adds this comment automatically for you. To understand how Apps Script represents your macro's instructions, you can look at the function: function Header(){ var spreadsheet = SpreadsheetApp.getActive(); var sheet = spreadsheet.getActiveSheet(); sheet.getRange(spreadsheet.getCurrentCell().getRow(), 1, 1, sheet.getMaxColumns()).activate(); spreadsheet.getActiveRangeList().setBackground('#4c1130') .setFontColor('#ffffff') .setFontWeight('bold'); spreadsheet.getActiveSheet().setFrozenRows(1); }; This code runs when you activate the Header macro. Following function, the label Header() defines the function's name and its parameters. Recognize that Header() requires no parameters as macro functions in Apps Script don't need inputs. The braces always enclose the body of a function in Apps Script. Later codelabs in this playlist explain the classes and concepts involved in creating the macro. For now, you can go through the following code descriptions to get a general idea of its components and their role in constructing your macro. Consider the first line: var spreadsheet = SpreadsheetApp.getActive(); Here, getActive() returns an object representing the current active spreadsheet file in Sheets and sets it to the new variable spreadsheet. var sheet = spreadsheet.getActiveSheet(); sheet.getRange(spreadsheet.getCurrentCell().getRow(), 1, 1, sheet.getMaxColumns()).activate(); These lines correspond to the action of clicking the first row to highlight it. This is called activation. The first line stores the current sheet in the variable sheet, while the second line gets the entire first row using the getRange() method and then calls activate() to activate it. The first row is specified using the specific row and column numbers. The spreadsheet.getCurrentCell().getRow() call returns the number of the current row, while sheet.getMaxColumns() returns the maximum number of columns in the sheet. spreadsheet.getActiveRangeList().setBackground('#4c1130') .setFontColor('#ffffff') .setFontWeight('bold'); This bit of code gets more complex. To efficiently call methods with spreadsheet, the code stacks three methods onto getActiveRangeList() to prevent the code from redundantly calling on this spreadsheet method more than once. As you code more using Apps Script, you'll get more familiar with this convention of calling multiple methods on one class (also known as method chaining). For now, you can read the following for brief explanations on each method in the code block: Lastly, the final line freezes the first row of the macro: spreadsheet.getActiveSheet().setFrozenRows(1); And that's the script you generated when you recorded your macro. Don't worry about any unfamiliar terms or methods mentioned above. The description is meant to get you thinking about some of the ideas Apps Script focuses on in a typical macro function, and the topics future codelabs delve into. The next section focuses on manipulating the Header() function's code to show how you can use the script editor to further personalize macros. Customize macros with Apps Script The Apps Script editor shows the macro you previously created in Google Sheets. By adjusting the contents of the function body, you can further customize your macro's instructions to take different or additional actions. The following exercises demonstrate various ways to manipulate macros with the script editor. Change the affected cells Suppose you want to modify your macro so it only affects the first 10 columns of the first row instead of the whole row. You could delete the macro and rerecord it. But, by using the Apps Script editor, you can make those changes directly. Here's one way of doing it: In the script editor, replace sheet.getMaxColumns() with 10. This edit changes the range of cells the macro affects in the spreadsheet. /** @OnlyCurrentDoc */ function Header(){ var spreadsheet = SpreadsheetApp.getActive(); var sheet = spreadsheet.getActiveSheet(); sheet.getRange(spreadsheet.getCurrentCell().getRow(), 1, 1, 10).activate(); /* sheet.getMaxColumns() replaced with 10.*/ spreadsheet.getActiveRangeList().setBackground('#4c1130') .setFontColor('#ffffff') .setFontWeight('bold'); spreadsheet.getActiveSheet().setFrozenRows(1); } To save your script, click Save . To rename your project, enter "Macros and Custom Functions" as the new project name and click Rename. To create a sheet, in Sheets, click Add Sheet . In the script editor, from the functions list, select Header and click Run. In your new sheet, you should see the following result: By modifying the active or target range, your macro now affects only part of the first row. Many Apps Script methods take a range or A1 notation as a parameter to specify which cells to act on. Next, let's learn about customizing your macro colors. Change the colors of your macro To help you design the color scheme of macros or other elements in Sheets, Apps Script can modify a range's fill or text color. Work through the following instructions to learn how you can customize the colors of your macro. These instructions focus on changing the background color of your macro: In Sheets, switch back to the sheet that contains the original data (Sheet 1). Click the first row to highlight it. In the script editor, replace background color #4c1130 with #afeeee. These values represent different colors using hex triplet notation. /** @OnlyCurrentDoc */ function Header(){ var spreadsheet = SpreadsheetApp.getActive(); var sheet = spreadsheet.getActiveSheet(); sheet.getRange(spreadsheet.getCurrentCell().getRow(), 1, 1, 10).activate(); spreadsheet.getActiveRangeList().setBackground('#afeeee') /* #4c1130 replaced with #afeeee.*/ .setFontColor('#ffffff') .setFontWeight('bold'); spreadsheet.getActiveSheet().setFrozenRows(1); } To save your script, click Save . From the functions list, select Header and click Run. In Sheets, the background fill of the first 10 columns in the first row recolors to a custom turquoise color: By switching the hex color notation in the parameters of setBackground(color) from #4c1130 (dark magenta 3) to #afeeee (pale turquoise, an option not accessible in Sheets' default color menu), you change the color attribute of your macro's background color. You've now modified the background color set by your macro. If you want to change the text color as well, change the second color code. In Sheets, click the first row to make sure it's still highlighted. In the script editor, replace font color #ffffff with #191970. This causes the macro to set a font color of navy blue. /** @OnlyCurrentDoc */ function Header(){ var spreadsheet = SpreadsheetApp.getActive(); var sheet = spreadsheet.getActiveSheet(); sheet.getRange(spreadsheet.getCurrentCell().getRow(), 1, 1, 10).activate(); spreadsheet.getActiveRangeList().setBackground('#afeeee') .setFontColor('#191970')/* #ffffff replaced with #191970.*/ .setFontWeight('bold'); spreadsheet.getActiveSheet().setFrozenRows(1); } To save your script, click Save . From the functions list, select Header and click Run. Return to Sheets to see the text color of the header row is now navy blue. Now you've seen how macros are actually Sheets actions recorded as Apps Script code. In the next section, you can view another way Apps Script can help you work with Google Sheets: custom functions. Like most spreadsheet applications, Google Sheets has several built-in formula functions such as =SUM() that enable quick calculations on spreadsheet data. Custom functions are simply functions you specify using Apps Script. Once you've defined a custom function, you can use it anywhere in your spreadsheet, just like a built-in function. This section shows you how to create a custom function in Apps Script that does a monetary conversion. Create a script file Using the same spreadsheet and script project from the Macros section, follow these instructions to learn how to make a new script (which you can eventually use to make your first custom function): To create an Apps Script file, return to the script editor. Beside Files, click Add a file > Script. Name the new script file customFunctions and press Enter. (Apps Script automatically appends a .gs extension to the script file name.) A new tab named customFunctions.gs appears within the editor. Now that you've created a script specifically for custom functions, you can fill it with code. Convert US dollars to Swiss francs Suppose you wanted to modify the data for 'Top 10 Highest Grossing Films 2018' to show not only the worldwide gross values in U.S. dollars but also in Swiss francs. With custom functions, you can do that easily. The following exercise demonstrates how to create a custom function to mathematically convert your dollar values to franc values. Before you can write your first custom function, modify your dataset to allow the function to demonstrate a proper output. To do this: In Sheets, right-click column H. In the resulting menu, click Insert 1 right. Label the column "Worldwide_Gross (Swiss francs)" in cell I1. Now you have a column that can store the results of your conversion custom function. Next, you can use the script editor to create your first custom function. In customFunctions.gs, replace the code for myFunction() with the following code: /** * Converts US dollars to Swiss francs. * * @param {number} dollars The total number of dollars. * @return {number} swissFrancs The converted total of Swiss francs. * @customfunction */ function USDTOCHF(dollars){ var swissFrancs = dollars * .99; return swissFrancs; } This is the code that will convert U.S. dollars to Swiss francs. Try the instructions below, and see how you can run a custom function in sheets. To save your script, click Save . In Sheets, select the I2 cell. In the function bar, enter =USDTOCHF(H2). To apply the formula to the rest of the cells in the column: Move your cursor to the bottom-right corner of the I2 cell and select the small blue box (your cursor should transform into when pointing to the blue box). Drag the blue box downwards to highlight the range I3:I11. Column I now lists the Swiss franc conversions of the US dollar values in Column H. Congrats, you've created your first custom function. The next section explains the code that comprises USDTOCHF(). Analysis of USDTOCHF() The initial comments detail the code's purpose: /** * Converts US dollars to Swiss francs. * * @param {number} dollars The total number of dollars. * @return {number} swissFrancs The provided value in Swiss francs. * @customfunction */ Comment blocks like this are used frequently in programming to explain what functions do. In this comment, you can identify two parts: the function description (to convert dollars to francs) and annotations that describe the function's parameters and return type. With the annotations, Apps Script utilizes JSDoc to help you document and create autocomplete hints for your code. You can read below how each annotation used in USDTOCHF() helps you with your Apps Script development: @param: You can use the @param annotation to describe each parameter passed into the function. @return: You can utilize the @return annotation to describe what the function returns. @customfunction: You should always add @customfunction in any custom function's doc comment. This annotation notifies Sheets to autocomplete your custom function just as Sheets autocompletes built-in functions when you enter a function name in a cell as seen below: Notice the text that appears in the autocomplete pop-up exactly matches the description text you placed in the comment block. You can make your custom functions easier to use by making sure the descriptions you create are well-written and complete. Next, focus on the code in the function USDTOCHF(): function USDTOCHF(dollars){ var swissFrancs = dollars * .99; return swissFrancs; } As mentioned previously, USDTOCHF() takes the numeric variable dollars, multiplies it by a fixed exchange rate, and returns a value converted to Swiss francs in the numeric variable swissFrancs. The input parameter is the value contained in the cell specified when adding the custom function to a cell. In this example, the input dollar amounts are coming from column H. The output value swissFrancs is placed in the function's cell (column I in this example). Custom functions can work with numeric or string values, as you'll see in the next section. Concatenate a string prefix Suppose you wanted the numeric output of the function USDTOCHF() to include the Swiss franc prefix CHF. You can do that with Apps Script by using the concatenation operator (+), as shown in the following instructions: In the script editor, update the @return annotation to return string instead of number. Change return swissFrancs to return 'CHF' + swissFrancs. The + operator appends the string CHF to the front of the value contained in swissFrancs. Your code should now look like this: /** * Converts US dollars to Swiss francs. * * @param {number} dollars The total number of dollars. * @return {string} swissFrancs The provided value in Swiss francs. * @customfunction */ function USDTOCHF(dollars){ var swissFrancs = dollars * .99; return 'CHF' + swissFrancs; } To save your script, click Save . The Swiss franc string now prefixes the values of column I: Your custom function now not only converts U.S. dollars to Swiss francs, but also outputs the currency with a string prefix. Advanced: Fetch external data This is a good start for a basic custom function, but this example assumes the exchange rate for dollars to Swiss francs is constant. Suppose instead you wanted to use the current exchange rate, so whenever the sheet is reloaded the values are recalculated to represent the current conversion? To do that, you'd need a means of discovering what the current exchange rate is. That's not information readily available in Google Sheets, but fortunately you can use Apps Script to get it. You can use code like below to get the current conversion rate of Swiss francs to U.S. dollars: function USDTOCHF(dollars){ // Gets a cache that is common to all users of the script. var cache = CacheService.getScriptCache(); // Accesses the memory location (rates.CHF) of the script cache. var rate = cache.get('rates.CHF'); // If a cache miss occurs, the program fetches the current // CHF rate from an API and stores the rate in the cache // for later convenience. if (!rate) { var response = UrlFetchApp.fetch(' var result = JSON.parse(response.getContentText()); rate = result.rates.CHF; cache.put('rates.CHF', rate); } // Converts dollars to CHF according to the latest rate. var swissFrancs = dollars * rate; // Returns the CHF value. return 'CHF' + swissFrancs; } This code fetches the current exchange rate from a financial information server using a third-party exchange rate API. This is done using Apps Script services like UrlFetchApp and CacheService. Those advanced concepts are out of scope for this specific codelab, but you can start to see the versatility of Apps Script to automate complex tasks in Google Sheets. Custom function guidelines Congratulations on completing the exercises for custom functions. As you use custom functions in your projects, it's important to understand they have certain restrictions. The following list summarizes the limitations detailed in the Custom Functions in Google Sheets guide: Don't create custom functions that require user authorization. Rather, build your custom functions to fulfill simpler tasks such as sample data calculations, text editing, etc. Go to Using Apps Script services. Don't name a custom function the same as another built-in function, or end the name with an underscore. Review the Naming guidelines. Don't pass variable arguments to custom functions. You can only pass deterministic (fixed) values to custom functions as arguments. Passing variable arguments, such as the result of =RAND(), will break the custom function. See the Arguments guidelines. Don't create functions that take more than 30 seconds to complete. If it takes longer an error will occur, so keep the function code simple and limited in scope. It's best to keep the calculations conducted in custom functions as simple as possible. View the Return values guidelines. Now you can improve your spreadsheets by using the script editor to work with macros and create custom functions. In the next section, you can review what you've learned, and what you can do next to improve your scripting skills. You've completed the first codelab of Fundamentals of Apps Script with Google Sheets. By creating and editing Sheets macros and custom functions, you've learned the basic Apps Script concepts. You can expand your Apps Script knowledge further in the next codelab. Yes No What we've covered Basic Apps Script concepts. How to navigate the script editor. How to create and update Sheets macros. How to create custom functions for Google Sheets. Key Terms Absolute reference: A macro recording setting specifying recorded macro actions are applied to the exact cells used in the recording. Active (status): Indicates the specified spreadsheet, sheet, range, or cell is the one the user of the spreadsheet is currently viewing or highlighting. Apps Script: A rapid application development platform, based on JavaScript, which makes it fast and easy to extend Google Workspace applications and automate work. Authorization: The process of the user granting permissions to allow Apps Script to access user data in the relevant Google services or otherwise take actions on behalf of the user. Container-bound script: Any script that is bound to and created from a Google Workspace document, such as a Google Sheet or Google Doc. Custom functions: User-defined formulas in Apps Script used in spreadsheets for simple operations. JSDoc: A markup language used to write explanatory notes on JavaScript-based source files. Macro: Short for "macroinstruction". A recorded series of Google Sheets actions you can repeat using a menu item or shortcut key. Method chaining: A programming idiom for invoking multiple method calls on an object, relying on the fact that each method call returns the object. Range: A group of one or more adjacent cells within a sheet. Relative reference: A macro recording setting specifying recorded macro actions are applied to the cells relative to the current cell selection. Script editor: The in-browser code editor for Apps Script. Sheet: A single page of a spreadsheet (similar to Microsoft Excel's workbooks). Sometimes used to refer to a Google Sheets file within Google Drive. SpreadsheetApp: This class serves as the parent class for the Spreadsheet service and provides a starting point for code that reads or manipulates Google Sheets data. Spreadsheet: A Google Sheets file that resides within Google Drive. Spreadsheet service: An Apps Script service that allows scripts to create, access, and modify Google Sheets files. Learn more about Apps Script in the Apps Script documentation. What's next The next codelab in this playlist presents the core classes and terminology of the Apps Script Spreadsheet service. This service enables you to finely control the values and presentation of data in Google Sheets using Apps Script. Find the next codelab at Spreadsheets, Sheets, and Ranges. [[["Easy to understand","easyToUnderstand","thumb-up"],["Solved my problem","solvedMyProblem","thumb-up"],["Other","otherUp","thumb-up"]],[["Missing the information I need","missingTheInformationINeed","thumb-down"],["Too complicated / too many steps","tooComplicatedTooManySteps","thumb-down"],["Out of date","outOfDate","thumb-down"],["Samples / code issue","samplesCodeIssue","thumb-down"],["Other","otherDown","thumb-down"]],[],[],[]] Have you ever wished your Google Sheets could do a little more heavy lifting for you? Like maybe automate some of those repetitive tasks or handle complex calculations without you having to lift a finger? Well, the good news is, you can! Google Sheets allows you to run scripts, which can help automate your workflows and enhance your spreadsheet experience in ways you might not have imagined. In this article, we're going to walk you through everything you need to know about running scripts in Google Sheets. From understanding what scripts are and how they work, to writing your own and integrating them with your spreadsheets, you'll be

equipped with the knowledge to take your Google Sheets skills to the next level. Let's get started!Build dashboards & reports in seconds with the best AI spreadsheet. Bricks makes creating dashboards, reports, and charts a breeze. Try it free → Before we dive into the how-tos, let's take a moment to understand what scripts are in the context of Google Sheets. Essentially, scripts are snippets of code that can perform tasks within your spreadsheet. These tasks can be as simple as automating formatting to as complex as pulling data from other sources. Google Apps Script is the scripting language that powers these scripts. It's based on JavaScript, so if you're familiar with that, you're already a step ahead. But even if you're not a coding whiz, don't worry. Google Apps Script is designed to be user-friendly, and with a bit of practice, you can start creating scripts that save you time and effort. Scripts can be used for a variety of purposes, such as: Automating repetitive tasks like formatting cells or generating reports. Integrating with other Google services like Gmail, Google Drive, or Google Calendar. Creating custom functions that extend the capabilities of Google Sheets. Pulling data from external APIs or databases. So, if you've ever found yourself doing the same thing over and over again in Google Sheets and thought, "There has to be a better way," scripts might just be your answer!Bricks is an AI-first spreadsheet that makes creating dashboards, reports, and analyzing your data a breeze. Import your data, describe what you want, and improve it by asking for changes.GET STARTED FOR FREE Okay, so you're ready to start scripting. But where do you begin? First, you'll need to access the Google Apps Script editor. Here's how to do it: Open your Google Sheet. Click on Extensions in the menu bar. Hover over Apps Script and click to open the script editor. Once you're in the editor, you'll notice it looks a lot like a code editor. Don't let that intimidate you—it's more approachable than it seems. The editor is where you'll write your scripts and connect them to your Google Sheet. Here's a simple example to get your feet wet. Let's say you want to create a script that automatically fills in a header row for you. You could write a script like this: function fillHeaderRow() { var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(); var header = ["Name", "Email", "Phone", "Address"]; sheet.appendRow(header); } To run this script, click the Run button in the toolbar. You'll likely need to authorize the script to make changes to your spreadsheet, so follow the prompts to do so. Once that's done, your script will execute and your header row will be added to the sheet! One of the coolest things about Google Apps Script is that it lets you create custom functions. If you've ever wished there was a formula that did exactly what you needed, now's your chance to create it. Custom functions in Google Sheets work just like built-in functions. You call them the same way, using an = sign, and they can take arguments just like any other function. Here's an example of a custom function that calculates the area of a rectangle: function RECTANGLEAREA(length, width) { return length * width; } To use this function, just type =RECTANGLEAREA(into a cell, followed by your length and width values. Hit enter, and voila! You've got your area. Creating custom functions can be a huge time-saver, especially if you find yourself doing the same calculations repeatedly. Plus, once you've written a function, you can reuse it as much as you want. Wouldn't it be nice if your scripts could run automatically without you having to click a button? That's where triggers come into play. Triggers are events that cause a script to run automatically. They're super handy for tasks that you want to automate fully. There are two types of triggers you can set up in Google Apps Script: Time-driven triggers: These run at specific intervals, like every hour or every day. Event-driven triggers: These run in response to specific actions, like opening a spreadsheet or editing cells. To set up a trigger, follow these steps: In the script editor, click on the clock icon to open the triggers menu. Click Add Trigger. Select the function you want to trigger. Choose the type of trigger (time-driven or event-driven). Configure the trigger settings according to your preferences. Once your trigger is set up, your script will run automatically based on the conditions you specified. It's like having a personal assistant who never forgets to do the repetitive stuff! Google Apps Script isn't limited to just Google Sheets. You can use it to integrate your sheets with other Google services, like Gmail, Google Drive, and Calendar. This opens up a world of possibilities for automating your workflows. For example, you can write a script that automatically sends an email when a certain condition is met in your spreadsheet. Or, you can create a script that moves files between Google Drive folders based on criteria you define. Here's a simple example of how you can send an email from Google Sheets: function sendEmail() { var emailAddress = "example@gmail.com"; var subject = "Hello from Google Sheets"; var message = "This is a test email sent from a script."; MailApp.sendEmail(emailAddress, subject, message); } With this script, you can automate sending emails without leaving your spreadsheet. Just imagine the possibilities! Even the best scripts can run into issues from time to time. Debugging and troubleshooting are important skills to ensure your scripts run smoothly. Here are some common strategies for finding and fixing script errors: First, use the Logger. Logger.log("This is a debug message"); Adding log messages in your code can help you track what's happening at various stages of your script's execution. You can view these logs by clicking on the View menu in the script editor and selecting Logs. Next, check for syntax errors. These are usually highlighted in red by the editor. Fixing these is often a straightforward process of checking your code for typos or missing punctuation. If your script isn't doing what you expect, it can be helpful to run it step by step and check the state of variables along the way. This can give you insights into where things might be going off track. Finally, don’t hesitate to search online. The Google Apps Script community is active and often has solutions to common problems. Platforms like Stack Overflow can be invaluable resources. Once you're comfortable with the basics, you might want to explore some advanced scripting techniques to enhance your Google Sheets experience even further. These techniques can help you create more efficient, powerful scripts that do exactly what you need. One technique is using external APIs. This allows your scripts to communicate with services outside of Google Sheets, bringing in data from the web into your spreadsheets. For instance, you could use an API to pull in weather data and update your sheet with the latest forecasts. Another advanced technique is using libraries. Libraries are pre-written code that you can include in your scripts to add functionality without having to write it from scratch. Google Apps Script offers several built-in libraries, and you can also create your own. Finally, consider learning more about JavaScript. Since Google Apps Script is based on JavaScript, improving your skills in this language can greatly enhance your scripting capabilities. There are countless resources online to help you learn, from tutorials to full-fledged courses. Once you've created a script you're proud of, you might want to share it with others. Google Apps Script makes it easy to do so by allowing you to publish your scripts as web apps or add-ons. To publish a script as a web app: Click on the Deploy button in the script editor. Select New deployment. Choose Web app as the deployment type. Fill in the necessary details and configure access permissions. Click Deploy. Your script will now be accessible via a URL, and others can use it as a web app. Similarly, you can publish your scripts as add-ons, which allows other users to install them directly into their Google Sheets. Sharing your scripts can be a great way to help others and even get feedback to improve your work. Plus, it's a chance to contribute to the community and maybe even gain a few insights yourself. To wrap up, let's look at some practical examples of scripts you might find useful in your daily work with Google Sheets. These examples are designed to spark ideas and show you just a few of the many ways you can use scripts to enhance your spreadsheets. First up, a script to automate data entry. If you have a form that collects data, you can write a script to automatically pull that data into your spreadsheet and organize it for you. function importFormData() { var formResponses = FormApp.getActiveForm().getResponses(); var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(); formResponses.forEach(function(response) { sheet.appendRow(response.getItemResponses().map(function(item) { return item.getResponse(); })); }); } Another useful script is one that generates charts based on your data. This can be especially helpful for creating visual summaries of your data at the click of a button. function createChart() { var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(); var range = sheet.getRange("A1:B10"); var chart = sheet.newChart() .setChartType(Charts.ChartType.BAR) .addRange(range) .setPosition(5, 5, 0, 0) .build(); sheet.insertChart(chart); } Finally, consider a script for sending reminders. If you have deadlines or meetings, you can automate reminders to ensure nothing slips through the cracks. function sendReminderEmails() { var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet(); var range = sheet.getDataRange(); var values = range.getValues(); values.forEach(function(row) { if (row[1] === "Upcoming") { // Assuming status is in the second column MailApp.sendEmail(row[2], "Reminder", "Don't forget your task!"); } }); } These examples only scratch the surface of what's possible with Google Apps Script. With a bit of creativity, you can tailor your scripts to fit your specific needs and transform the way you work with Google Sheets.Bricks makes it easy to analyze data, create dashboards and reports, and get insights from your spreadsheet data.SIGN UP for free We've covered a lot of ground, from the basics of what scripts are to writing custom functions and automating tasks with triggers. Hopefully, you're feeling inspired to try your hand at scripting in Google Sheets and see how it can make your work a little easier and a lot more efficient. But what if you're looking for a tool that takes things even further? That's where Bricks comes in. Bricks integrates spreadsheets, docs, and presentations into one seamless tool, with AI at its core. It can do everything in the spreadsheet for you—like write your formulas, clean data, or create charts and graphics. Essentially, anything you can do in a spreadsheet, Bricks AI can do it for you in seconds, so you don't have to be a spreadsheet expert. Plus, it can create visuals based on your spreadsheet data, such as dashboards, reports, charts, graphs, calendars, timelines, project trackers, org charts, roadmaps, and so much more. And it's all connected with your docs and presentations. So, if you're ready to supercharge your productivity, give Bricks a try!header-1-2header-1-4header-2-4header-3-4header-4-4header-5-4header-1-6header-2-6header-3-6header-1-8Bricks is the AI-first dashboard and reporting tool for spreadsheet data. Create charts, graphs, and analyze your data in seconds - no data analyst needed.Create your first report