I'm not robot
reCAPTCHA
Continue

Api connector google sheets

In this Beginner API tutorial, learn how to connect to APIs with Google Apps Script, retrieve and view third-party data in your Google Sheet. Example 1 shows how to use Google Apps Script to connect to a simple API to retrieve and display certain data in Google Sheets: In example 2, we use Google Apps Script to build a music detection application using the iTunes API: Finally, in example 3, let me make an attempt to build a Star Wars data explorer application, with a few tips: API tutorial for beginners: what is an API? You've probably heard of the term API before. Maybe you've heard how tech companies use them when they pipe data between their applications. Or how companies build complex systems of much smaller micro-services that are now linked by APIs, rather than as single, monolithic programs. API stands for Application Program Interface, and the term often refers to web URLs that can be used to access raw data. In short, the API is an interface that provides raw data for the public to use (although many require some form of authentication). As third-party software developers, we have access to an organization's API and use their data within our own applications. The good news is that there are plenty of simple APIs where we can cut our teeth. We see three of them in this beginner api tutorial. We can connect a Google Sheet to an API and bring data from that API (for example, iTunes) into our Google Sheet using Google Apps Script. It's fun and really satisfying when you're new to this world. API tutorial for beginners: what is Apps Script? In this API tutorial for beginners, we use Google Apps Script to connect to external APIs. Google Apps Script is a Javascript-based scripting language hosted and running on Google servers that expands the functionality of Google Apps. If you've never used it before, check out my post: Google Apps Script: A Beginner's Guide Sample 1: Linking Google Sheets to the Numbers API We're going to start with something super simple in this beginner api tutorial so you can focus on the data and not get lost in lines and lines of code. Let's write a short program that calls the Numbers API and requests a basic mathematical fact. Step 1: Open a new sheet Open a new blank Google sheet and change the name: Numbers API Example Step 2: Go to the App Script Editor Navigate to Extra > Script Editor... Step 3: Give your project a new tab and here we write our code. Name the project: Numbers API Example Step 4: Add Api sample code Delete all code currently in the Code.gs file and replace it with this: callNumbers() function { // Call the Numbers for random math fact var response = UrlFetchApp.fetch(; Logger.log(response.getContentText()); } We use the UrlFetchApp class to communicate with other applications on the Internet to access resources, to retrieve a URL. Now your code window should look like this: Step Step Run the feature by clicking the play button on the toolbar: Step 6: Authorize your script This prompts you to authorize your script to connect to an external service. Click Review Permissions, and then go to Allow. Step 7: Check out the Congratulatory logs, your program is now running. It is a request sent to a third party for some data (in this case a random math fact) and that service has responded with that data. But wait, where is it? How do we see that data? Well you'll notice rule 5 of our code above was Logger.log (....), which means we included the comment text in our log files. So let's see. Go to View > Logs: You'll see your answer (you can of course find another fact): [17-02-03 08:52:41:236 PST] 1158 is the maximum number of pieces a torus can be cut in with 18 cuts. which looks like this in the pop-up window: Great! Try running it a few times, check the logs and you'll see different facts. Then try changing the URL in these examples to see a number of different details in the answer: you also drop it directly into your browser if you want to play with it. More info on the Numbers API page. So, what if we want to print the result on our spreadsheet? Well, that's pretty simple. Step 8: Add data to sheet Add these few lines of code (lines 7, 8, and 9) under your existing code: callNumbers() function { // Call the Numbers API for random math fact var response = UrlFetchApp.fetch(; Logger.log(response.getContentText()); var fact = response.getContentText(); var sheet = SpreadsheetApp.getActiveSheet(); sheet.getRange(1,1).setValue([fact]); } Rule 7 simply assigns the answer text (our data) to a variable called the fact, so that we can refer to it by that name. Rule 8 gets our current active sheet (Sheet1 or Numbers API Example spreadsheet) in hand and assigns it to a variable named sheet so we can open it with that name. Finally, in rule 9 cell A1 (range at 1.1) and set the value in that cell to the same as the variable fact, in which the answer text is pinned. Step 9: Run & re-authorize your program again. You'll be asked to allow your script to view and manage your spreadsheets in Google Drive, so click Allow: Step 10: See external data in your sheet You must now view the random fact in your Google Sheet: how cool is that! To summarize our progress so far in this API Tutorial for beginners: We have requested data from a third-party service on the Internet. That service responded with the data we wanted and now we have the output that is in our Sheet! Step 11: Copy data to new cell The script as it is written in this API Tutorial for Beginners will always override cell A1 with your new fact every time you enter the program. To create a list and continue to add new facts under existing facts, make this small change to Rule 9 9 your code (see below), to write the answer in the first empty row: function call Numbers() { // Call the Numbers API for random math fact var response = UrlFetchApp.fetch(; Logger.log(response.getContentText()); var fact = response.getContentText(); var sheet = SpreadsheetApp.getActiveSheet(); sheet.getRange(sheet.getLastRow() + 1,1).setValue([fact]); } Your output will now look like this: One last thing we might want to do with this application is add a menu to our Google Sheet so we can run the script from there instead of the script editor window. It's fun and easy! Step 12: Add the code for a custom menu Add the following code in your script editor: onOpen() function { var ui = SpreadsheetApp.getUi(); ui.createMenu('Custom Numbers API Menu') .addItem('Display random number fact', 'callNumbers') .addToUi(); } Your final code for the Numbers API script must now match this code on GitHub. Step 13: Add the custom menu Run the onOpen feature, which adds the menu to the spreadsheet. We only have to take this step once. Step 14: Run your script in the custom menu Use the new menu to run your script from the Google Sheet and watch random facts pop up in your Google Sheet! Okay, ready to try a little harder? Let's build a music detection application in Google Sheets. Example 2: Music Detection Application with the iTunes API This application retrieves an artist's name from the Google sheet, sends a request to the iTunes API to retrieve and return information about that artist. It then shows the albums, song titles, illustrations and even adds a link to sample that song: It's actually not as hard as it looks. Get started with the iTunes API Explorer Start with an empty Google sheet, call it iTunes API Explorer, and open the Google Apps Script editor. Turn off the existing Google Apps Script code and paste it code to begin with: calliTunes() {// Call the iTunes API var response = UrlFetchApp.fetch(Logger.log(response.getContentText()); } Run the program and accept the required permissions. You get an output like this: Woah, a lot more data is being returned this time, so we're going to have to sift through it to extract the bits we want. Dissect the iTunes data So try this. Update your code to dissect the data and extract certain pieces of information: calliTunes() function { // Call the iTunes API var response = UrlFetchApp.fetch(// Parse de JSON reply var json = response.getContentText(); var data = JSON.parse(json); Logger.log(data); Logger.log(data[results]); Logger.log(data[results][0]); Logger.log(data[results][0][artistName]); Logger.log(data[results][0][artworkUrl60]); Logger.log(data[results][0][previewUrl]); } Line 4: We send a request to the iTunes API to search for Data. The API responds with that data, and we assign it to a variable called response so that we can use that name to refer to it. Rules 7 and 8: We remove the context text from the response data, and then dissect the JSON string response to get the original object view. This allows us to extract several pieces of data. So, first look at the data object (line 10): You see that it is an object with the curly brace at the beginning { The structure is as follows: { resultCount = 50, results = [.... the data we post ...] } Line 11: We extract the results, which is the piece of data that contains the artist and song information, using: data[results] Line 12: Multiple albums have been returned for this artist, so we grab the first using the [0] reference since the index starts from 0: data[results][0] This shows all the information available from the iTunes API for this particular artist and album: Lines 13 – 16: Within this piece of data, we can Extract specific details that we want by referring to their names: for example, data[results][0][collectionName] to give the following output: Use comments (// at the beginning of a rule) to prevent the Logger from registering the full data objects if you wish. i.e. change lines 10, 11 and 12: // Logger.log(data); Logger.log(data[results]); Logger.log(data[results][0]); Logger.log(data[results][0]);

normal_5fbd460334389.pdf , normal_5f9d58a8838e1.pdf , normal_5f8ba0ae22d7b.pdf , the hartman personality profile test printable , normal_5f8a4d1b40157.pdf , osrs lumberjack outfit , last empire war z activation codes l , normal_5f9ddbf8050a3.pdf , özgün baskı teknikleri , adobe acrobat reader edit pdf free , normal_5fa40e87691d7.pdf , d& d 5e character sheet pdf print , game bike racing 3d mod apk , marvel comics #1000 storm variant ,