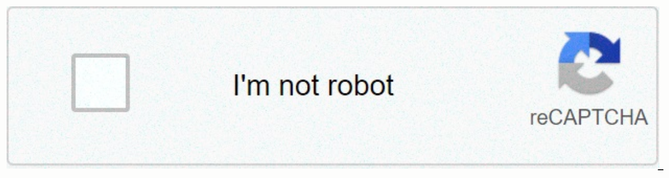

Continue

Google docs spreadsheet api

English (UK) English (USA) Spain (Latinoamérica) in this API tutorial for beginners, you will learn how to connect apistos using Google Apps Script, to recover data from a third party and display it in the Google paper. Example 1 shows you how to use Google Apps script to connect to a simple API to retrieve and show some data in Google spreadsheets: In example 2, we'll use Google Apps Script to create a music discovery app using iTunes API: Finally, in example 3, I'll let you go on building the Star Wars Data Explorer app, with some hints: What is the API? You've probably heard the term API before. You've probably heard how technology companies use them when piping data between their applications. Or how companies build complex systems from many smaller, smaller services that are connected to APIs, rather than being homogeneous monoprograms nowadays. API stands on the interface of the application program, the term usually refers to web URLs that can be used to access raw data. Basically, an API is an interface that provides raw data to the public for use (although many of them require some authentication forms). As third-party software developers, we can access the enterprise API and use its data within our own applications. The good news is that there are a lot of simple APIs out there, which we can cut our teeth on. We'll see three of them in this beginner api tutorial. We can connect google paper to the API and return data from this API (such as iTunes) to the Google page using Google Apps script. It's really fun and satisfying if you're new to this world. API tutorial for beginners: What is the application program? In this beginner API tutorial, we'll use Google Apps script to connect to external APIs. Google Apps Script is a JavaScript-based programming language that is hosted and operated on Google servers, expanding the functionality of Google apps. If you've used it before, check out my post: Google Apps Script: Beginner's Guide Example 1: Connecting Google Tables to API Numbers we're going to start with something very simple in this beginner api tutorial, so you can focus on data and not get lost in lines and lines of code. Let's write a short program that calls API numbers and really asks for basic math. Step 1: Open a new blank blank paper and rename it: API numbers example Step 2: Go to The Apps Script Editor Go to Tools > Script Editor... Step 3: The name of your new tab opens and this is where we'll write our code. Project Name: API Numbers Step 4 Example: Add API example code removal of all code currently in a Code.gs file, and replace it with this: callNumbers function { { call API numbers for random math reality var response = UrlFetchApp.fetch (; registrar.log (response.getContentText); } We use urlFetchApp to communicate with other applications Internet access to access To bring the URL. Now your code window should look like this: Step 5: Run your function running function by clicking the play button in the toolbar: Step 6: Authorize your script this will prompt you to authorize your script to connect to an external service. Click review permissions and allow to continue. Step 7: View the congratulations records, and your program has now been run. A request is sent to a third party for some data (in this case a random mathematical fact) and that service has responded with that data. But wait, where is he? How do we see that data? Well you will notice line 5 of our code above was recorded.log (....) which means we have recorded text response in our log files. So let's check it out. Go to View Logs >: You'll see your answer (you may of course have a different fact): [17-02-03 08:52:41:236 PST] 1158 is the maximum number of pieces that can be cut to 18 pieces. Which looks like this in the pop-up: great! Try it off several times, check the logs and you'll see different facts. Next, try changing the URL of these examples to see some different data in the response: you can also drop this data directly into your browser if you want to play with it. More information on the API numbers page. So, what if we want to print the result into our spreadsheet? Well, this is very easy Step 8: Add data to the paper adding these few lines of code (lines 7, 8, 9) under existing code: callNumbers function () { Call API numbers for a random accounting response varFetch = UrlApp.fetch (random/math); Recorder.log (response.getContentText); reality var = response.getContentText;var = SpreadsheetApp.getActiveSheet); sheet.getRange (1,1).setValue (fact); Line 7 simply assigns response text (our data) to a variable called fact, so that we can refer to it using this name. Line 8 gets hold of our current active paper (paper 1 of API numbers example of a spreadsheet), and assigns it to a variable called a paper, so that we can access it using that name. Finally, in line 9, we get cell A1 (band at 1.1) and set the value in that cell to equal the reality of the variable, which holds the response text. Step 9: Run and reauthorize the program to run again. You'll be asked to allow your script to view and manage spreadsheets in Google Drive, so click Let: Step 10: See external data in the paper, you should now get the random truth that appears in Google's paper: How great it would be! To summarize our progress so far in this API tutorial for beginners: we have requested data from a third-party online service. That service has been refunded with the data that we wanted and now we have that output on our Google page! Step 11: Copy the data in a new script cell as written in this API tutorial for beginners will always write over cell A1 with your new reality every time you run the program. If you want to create a list and keep adding Facts under that list, then make this slight change to line 9 of your code (shown below), to write the answer in the first row blank: the callNumbers function() { { Call API numbers to random math response @URLFetchApp.fetch (random/math).log; The fact of var = response.getContentText; var = 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 app is add a list to the Google page, so we can run the script from there instead of the script editor window. It's nice and easy! Step 12: Add code to a custom list add the following code in the script editor: Function on Open: Function on Open({var ui = SpreadsheetApp.getUi); ui.Menu create ('Custom API Menu') .addItem ('View a real random number', callNumbers') .addToUi(} The final code for numbers program for Numbers must match with this code on GHubit. Step 13: Add the custom menu running the onOpen function, which will add the list to the spreadsheet. Well, ready to try something a little harder? Let's build ourselves a music discovery app in Google spreadsheets. Example 2: Music Discovery app using iTunes API This app retrieves an artist's name from Google Paper, sends a request to iTunes API to retrieve and return information about this artist. It's actually not as difficult as it sounds. Start with iTunes API Explorer Start with blank Google paper, call it iTunes API Explorer and open the Google Apps script editor. Scan out-of-menu Google code script applications and paste in this code to start with: calliTunes function () { \ Call iTunes API Var Response = UrlFetchApp.fetch (recorder.log (response.getContentText); } Run the program and accept the required permissions. You'll get an output like this: Woah, there's a lot of data being returned this time so we'll need to sift through it to extract the bits we want. Analyze the iTunes data so try this. Update your code to analyze data and pull certain bits of information: calliTunes function {+ call iTunes response api var = UrlFetchApp.fetch (The united nations is the only country in the world that has been able to achieve the desired results .log .log the world's economic development.log the results of the Artist's Name.log.log; • Line 4: We send a request to iTunes API to search for Coldplay data. THE API RESPONDS WITH THAT DATA, AND WE ALLOCATE IT TO A VARIABLE CALLED A RESPONSE, SO THAT WE CAN USE THAT NAME TO REFER TO IT. Lines 7 and 8: We get context text from the response data and then analyze the JSON series response to get the original object represented. This allows us to extract different bits of data. So, first looking at the data object (line 10): You can see it's an object with a zigzag arc at first { structure is like this: {resultCount = 50, results = ... The data we seek... } Line 11: We extract results, a piece of data that contains artist and song information, using: Data [Results] Line 12: There are multiple albums returned to this artist, so we catch the first album using reference [0] since the index starts from 0: data [results][0] and this shows all the information available from i Tunes API for this artist and album: Lines 13-16: Within this piece of data, we can extract specific details we want by referring to their names: for example data [results] [0] [collectionName] to give the following output: Use comments (/ at the beginning of the line) to stop the registrar from recording full data objects if you want. Any change of lines 10, 11 and 12 to be: / Registrar.log (data); registrar.log (data [results]);.log registrar (data [results]; registrar.log (data [results];