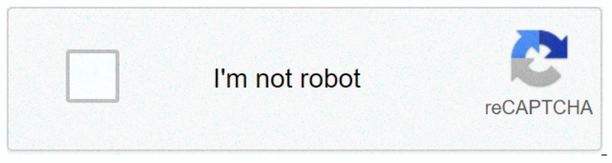

Continue

Google sheets api example

type: thumb-down, id: missingTheInformationINeed, label:Missing the information I need , type: thumb-down, id: tooComplicatedTooManySteps, label:Too complicated / too many steps -down, id: outOfDate, label:Out of date , type: thumb-down, id: samplesCodeIssue, label:Samples/Code issue [type: thumb-up, id: easyToUnderstand, label:Easy to understand , type: thumb-up, id: solvedMyProblem, label:Solved my problem The Google Sheets API allows you to read and edit any aspect of a spreadsheet. Spreadsheets have a lot of settings that allow you to create beautiful and functional sheets, so the API also has a lot of settings. The API offers two main ways to interact with the spreadsheet: both of these collections are simple to use, although spreadsheet collection provides many other options. For more information about these libraries and how to use them, see links to the previous reference documentation or the following developer guides: Reading and writing values that update spreadsheets This section describes some of the terms you encounter when reading the Sheets API. Spreadsheet ID Each API method requires a spreadsheetId parameter that is used to identify the spreadsheet to access or modify. This ID is the value between /d/ and /edit in the SPREADSHEET URL. For example, consider the following URL that references a GoogleSheet spreadsheet: 0 Spreadsheet ID is a string that contains letters, numbers, and some special characters. The following regular expression can be used to extract the spreadsheet ID from a Google Sheets URL: /spreadsheets/d/([a-zA-'0-9-_]) If you are familiar with the Drive API, the spreadsheetId is the File resource ID. Sheet ID Individual sheets in a spreadsheet have titles (which must be unique) and IDs. The sheetId is frequently used in the Sheets API to specify which sheet is read or updated. In the Sheets UI, you can find the sheetId of the open sheet in the SPREADSHEET URL, as the value of the gid parameter. The following is the structure of the URL and where sheetId can be found: sheetId The sheet ID is numeric, and the following regular expression can be used to extract it from a Google Sheet URL: [&]gid-([0-9]) You can also retrieve sheet IDs using the API - see the Determine sheet ID sample and other properties. A1 notation Some API methods require a range in A1 notation. This is a string like Sheet1! A1:B2, which references a group of cells in the spreadsheet and is typically used in formulas. For example, valid ranges are: Sheet1! A1: B2 refers to first two cells in the first two rows of Sheet1. Sheet1! A:A refers to all cells in the first column of Sheet1. Sheet1!1:2 refers to all cells in the first two rows of Sheet1. Sheet1! A5:A refers to all cells in the first column of sheet 1, from row 5 onwards. A1: B2 refers to the first two cells in the first two rows of the first visible sheet. Sheet1 refers to all cells on Sheet1. Named ranges are also supported. When a named range conflicts with a sheet name, the named range is preferable. If the sheet name contains spaces or starts with a bracket, enclose the sheet name in single quotation marks ('), such as 'Sheet One'! A1:B2. For simplicity, it is safe to always surround the sheet name with single quotation marks. Google sheets, like most other spreadsheet applications, treat date/time values as decimal values. This allows you to perform arithmetic operations on them in formulas, so you can increment days or weeks, add or subtract two dates/hours, and do other similar things. Google sheets uses a form of vintage date commonly used in spreadsheets. The part of the integer of the value (to the left of the decimal) counts the days from December 30, 1899. The fractional part (to the right of the decimal) counts the time as a fraction of a day. For example, January 1, 1900 at noon would be 2.5, 2 because it's two days after December 30, 1899, and .5 because noon is half a day. February 1, 1900 at 3:00 pm would be 33,625. Keep in mind that Google Sheets correctly considers the year 1900 as a common year, not as a leap year. When you read cell values, you can get rendered dates as strings instead of serial values, using spreadsheet.values.get together with the DateTimeRenderOption. Partial answers Spreadsheets are large and you often don't need every part of the spreadsheet. You can limit what is returned in a response to the Google Sheets API, using the URL parameter of the fields. This is especially useful in the spreadsheets.get method. For best performance, explicitly list only the fields you need in the response. The format of the fields parameter is the same as the JSON encoding of a FieldMask object. Simply put, multiple different fields are separated by commas, and subfields are separated by dots. For convenience, multiple subfields of the same type can be listed in parentheses. For example, to retrieve the spreadsheet title, sheet properties, and the value and format of the range A1:C10. You can use the following request: GET In this API tutorial for beginners, you'll learn how to connect to using Google Apps Script, to retrieve data from a third party and display it in your Google sheet. Example 1 shows how to use Google Apps Script to connect to a simple API to retrieve some data and show it in Google Sheets: in Example 2 we'll use Google Apps Script to create a music discovery application using the iTunes API: Finally, in Example 3, I'll let you go and build a Wars Data Explorer application, with some tips: API tutorials for beginners: what is an API? You've probably heard the term API before. You may have heard how tech companies use them when pipe data between their applications. Or how companies build complex systems from many smaller microservians connected by APIs, rather than as individual monolithic programs nowadays. API stands for Application Program Interface, and the term commonly refers to Web URLs that can be used to access raw data. Basically, 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 can access an organization's API and use their data within our applications. The good news is that there are a lot of simple APIs out there, that we can cut our teeth on. We will see three of them in this beginner bee tutorial. We can link a Google sheet to an API and report data from that API (e.g. iTunes) to our Google sheet using Google Apps Script. It's fun and really satisfying if you're new to this world. Beginner bee tutorial: What is Apps Script? In this BEGINNER API tutorial, we'll use Google Apps Script to connect to external APIs. Google Apps Script is a Javascript-based scripting language hosted and run on Google servers, extending the functionality of Google Apps. If you've never used it before, check out my post: Google Apps Script: A Beginner's Guide Example 1: Connecting Google Sheets to the Numbers API We'll 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 requires a basic mathematical fact. Step 1: Open a new blank Google sheet Open a new blank Google sheet and rename it: Numbers API Example Step 2: Go to application script editor Go to Tools > Script Editor... Step 3: Name the project A new tab opens and this is where we will write our code. Name the project: Numbers API Example Step 4: Add API sample code Remove all code that is currently in the Code.gs file and replace it with the following: function callNumbers() / / Call the Numbers API for random math fact var response - UrlFetchApp.fetch(; Logger.log(response.getContentText()); We are using the UrlFetchApp class to communicate with other applications on the Internet to access resources, to retrieve a URL. Now the code window should look like this: Step 5: Run the Function by clicking the play button toolbar: Step 6: Authorize the script This will ask you to authorize the script to connect to an external service. Click Check Permissions, and then click Allow to continue. Step 7: View congratulations logs, the program is now running. A request is sent to a third party for some data (in this case a random mathematical fact) and that service responded with the data. But wait, where is he? As As let's see that data? Well, you'll notice line 5 of our code above was Logger.log(....) which means we recorded the response text in our log files. So let's take a look. Go to View > Logs: You'll see your answer (you can of course have a different fact): [17-02-03 08:52:41:236 PST] 1158 is the maximum number of pieces a bull can be cut with 18 cuts. that looks like this in the popup window: Big! Try to run a couple of times, check the logs and you will see different facts. Next, try changing the URL in these examples to see some different data in the response: You can also drop it directly into your browser if you want to play with them. Learn more on the Numbers API page. So, what if we want to print the result on our spreadsheet? Well, it's pretty easy. Step 8: Add the data to the sheet Add these few lines of code (lines 7, 8, and 9) under the existing code: callNumbers() / / / Call the Numbers API for the math fact var random response - UrlFetchApp.fetch(; Logger.log(response.getContentText()); var fact : response.getContentText(); var sheet : SpreadsheetApp.getActiveSheet(); sheet.getRange(1,1).setValue([fact]); Line 7 simply assigns the response text (our data) to a variable called fact, so you can refer to it using that name. Row 8 contains the current active sheet (Sheet1 of the Sheet Example API number sheet) and assigns it to a variable called a sheet, so that it can be accessed using that name. Finally, in line 9, we get cell A1 (range to 1.1) and set the value in that cell to be equal to the variable fact, which contains the response text. Step 9: Run and re-authorize Run the program again. You'll be asked to allow your script to view and manage spreadsheets in Google Drive, then click Allow: Step 10: See external data in your sheet You should now get the random fact that appears in your Google sheet: How good it is! To summarize our progress so far in this BEGINNER API tutorial: We requested data from a third-party service on the Internet. This service responded with the data we wanted and now we have output that in our Google sheet! Step 11: Copy the data into the new cell The script as it is written will always overwrite cell A1 with the new fact each time you run the program. If you want to create a list and continue adding new facts to existing ones, make this minor change to line 9 of the code (shown below) to write the response in the first empty line: function callNumbers() - // Call the API for math fact var random 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 Google Sheet, so you can run the script from there rather than the script editor window. It's nice and easy! Step 12: Add code for a custom menu Add the following code in the script editor: function onOpen() - var ui - SpreadsheetApp.getUi(); ui.createMenu('Custom Numbers API Menu') .addItem('View fact random number','callNumbers') .addToUi(); The final code for the Numbers API script should now match this code in GitHub. Step 13: Add the custom menu Run the onOpen function, which will add the menu to the spreadsheet. We only have to make this step once. Step 14: Run your script from the custom menu Use the new menu to run the script from the Google sheet and watch random pop-up facts in your Google sheet! All right, ready to try something a little harder? Let's build a music discovery application in Google Sheets. Example 2: Music Discovery Application using the iTunes API This application retrieves an artist's name from the Google sheet, sends a request to the iTunes API to retrieve information about that artist, and return it. Then view albums, song titles, graphics, and also add a link to sample that track: it's actually not as difficult as it sounds. To get started with iTunes API Explorer Start with a blank Google sheet, name it iTunes API Explorer, and open the Google Apps script editor. Clear the existing Google Apps script code and paste into this code to start with: calliTunes() / / / Call the iTunes API var response - UrlFetchApp.fetch(Logger.log(response.getContentText()); Run the program and accept the necessary permissions. You will get an output like this: Woah, there is a lot more data returned this time, so we will need to sift through it to extract the bits we want. iTunes data analysis So try this. Update your code to analyze data and extract some bits of information: calliTunes() function - // Call the iTunes var response API - UrlFetchApp.fetch(Parse the 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][propertyname]); 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 Coldplay data. The API responds with data and we assign it to a variable called response, so you can use that name to reference it. Lines 7 and 8: You get context text from the response data, and then parse the JSON string response to get the object representation This allows us to extract different bits of data. So, looking first at the data object (row 10): you can see that it is an object with the brace at the beginning : the structure is similar to this: resultCount - 50, results [[data we're looking for...] Line 11: Let's extract the results, which is the piece of data that contains the artist's and song's information, using: data[results] Line 12: There are multiple albums returned for this artist, so let's take the first one using the reference [0] since the index starts at 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 provide the following output: Use comments (// at the beginning of a row) to prevent the Logger from registering full data objects if desired. For example, change lines 10, 11, and 12 so that: // Logger.log(date); Logger.log(data[results]); Logger.log(data[results][0]); Logger.log(data[results][0]);