

JavaScript

Intro for Beginners

Lab 4

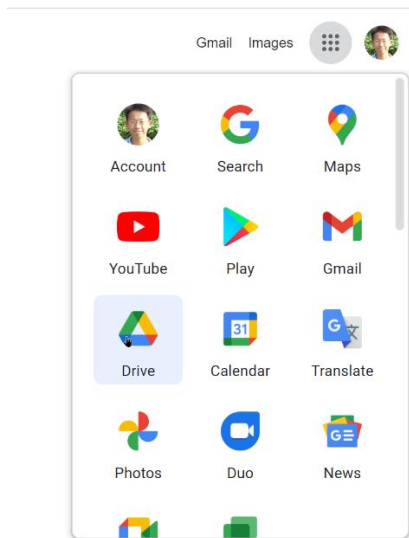
1	BASIC INTRO TO GOOGLE APPS SCRIPT WITH GOOGLE SHEET	1
1.1	CREATING YOUR FIRST APPS SCRIPT	1
1.2	CALLING CUSTOM FUNCTIONS FROM GOOGLE SHEET.....	4
1.3	READING AND WRITING DATA FROM AND TO A CELL RANGE IN GOOGLE SHEET	6
1.4	GENERAL APPROACH TO WORKING WITH GOOGLE APPS SCRIPT.....	8
2	TRY - CATCH - FINALLY	9
3	PROMISES	9
4	WORKING WITH JSON	9
5	USING THE FETCH API	10
6	IMPLEMENTING A SIMPLE API SERVER WITH EXPRESS.JS	11

1 Basic intro to Google Apps Script with Google Sheet

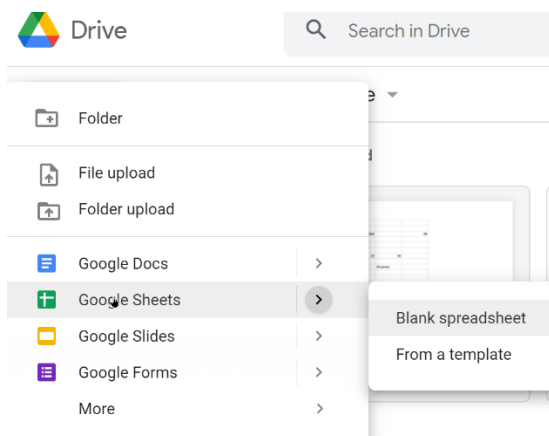
1.1 Creating your first Apps Script

<https://spreadsheet.dev/creating-your-first-apps-script>

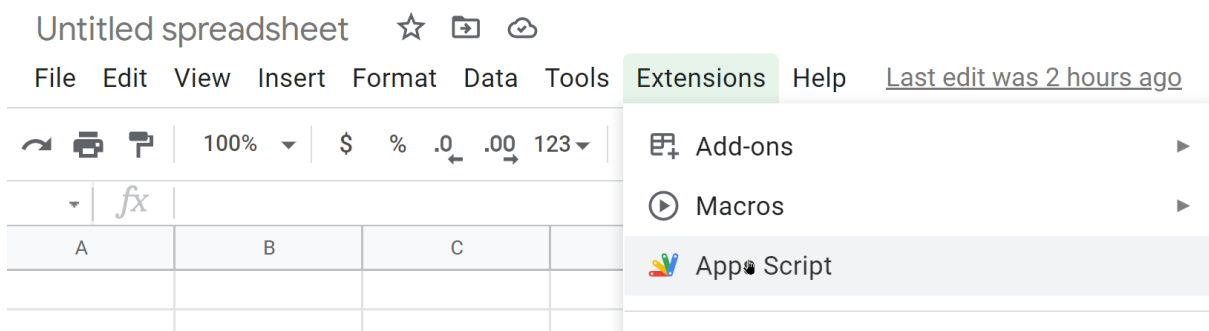
Make sure you are logged into your Gmail account and open a new browser tab in Chrome. Select the Drive icon from the drop down menu in the upper right hand corner.



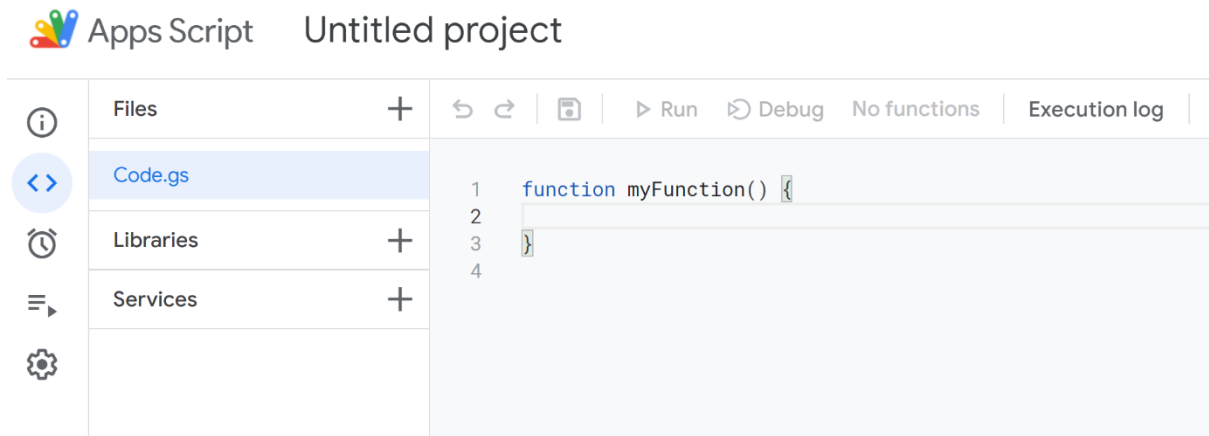
From the Drive menu panel on the left, select New -> Google Sheets to create a new Google Sheet.



Select the Apps Script Editor option from the Extensions option in the main menu at the top.

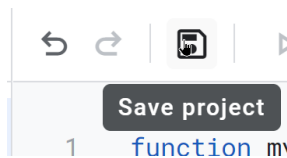


This opens up the Google Apps Script Editor view with a new untitled project and an empty function

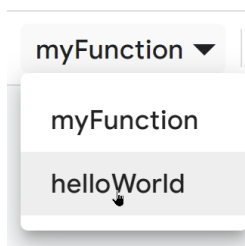


File to use: gappscript-intro.js

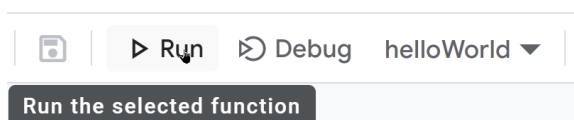
Copy the function `helloWorld` and paste it anywhere below the predefined empty function `myFunction`. Click **Save Project** (remember that you must always save your project before you run it).



From the drop down menu for functions, select `helloWorld`.



Always make sure you have the correct function selected before you attempt to run. Next click **Run** to run the selected function.



Executing the program may take a while to complete because although you are typing the code into your browser, the actual execution of the code is performed by a server in Google's cloud platform. The final result in terms of the output from the various `Logger.log` statements are shown in the Execution Log at the bottom. You can toggle the display of this log using the button at the top.

Execution log

9:39:59 AM	Notice	Execution started
9:40:02 AM	Info	Hello World !
9:40:02 AM	Info	This is my first Google Apps Script function
9:40:02 AM	Info	3 + 5 gives us 8
9:39:59 AM	Notice	Execution completed

Make a few more modifications to the `helloWorld` function by adding in additional `Logger.log` statements or changing the messages to be displayed in the existing statements. You can also create a few more variables and basic mathematical expressions and output the result in a similar manner.

Always remember to save your changes before clicking on the Run button to execute the selected function.

Copy the function `demoSampleStuff` and paste it the empty space at the bottom of the editor view and save the project (remember that you must always save your project before you run it). Run it and verify that output.

You can go through the tutorial below to revise and apply the various JavaScript constructs you have learned previously within the context of a Google App Script function.

<https://spreadsheet.dev/learn-coding-google-sheets-apps-script>

1.2 Calling custom functions from Google Sheet

File to use: `gappscript-intro.js`

Copy the function `addThreeNumbers` and paste it the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

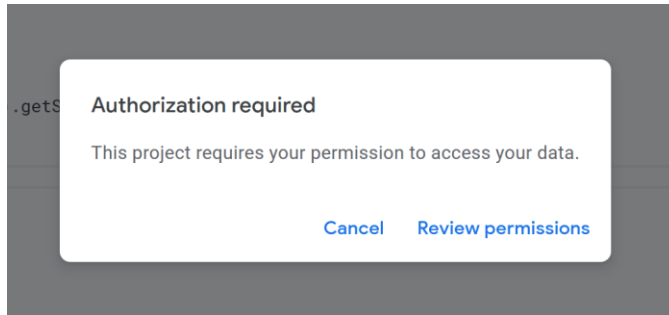
Back in Google Sheet, enter three random numbers into any 3 cells at the upper right hand corner of the current active sheet. Then, in another empty cell call this function via the function bar at the top and pass it the coordinates of the 3 cells containing these random numbers and press enter.

	B	C	D	E
	10	20	30	

The empty cell should now hold the value returned from the function `addThreeNumbers`

If you see any message related to not being able to locate this function in Google Sheet, check that you have typed the name of the function correctly and refresh the sheet in the browser tab with F5.

If you see any message related to authorization, such as that shown below:



Follow the sequence of steps outlined in:

<https://spreadsheet.dev/authorizing-an-apps-script>

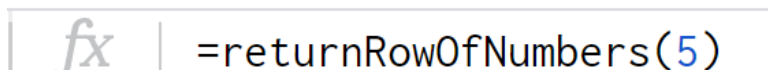
Try entering new values into those 3 cells, and you will notice the function runs immediately with every change to return the latest updated result.

<https://spreadsheet.dev/returning-values-from-custom-functions-in-google-sheets>

In addition to returning a single value, we can return a row or column of values and a table (2-dimensional array) of values.

Copy the function `returnRowOfNumbers` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Back in Google Sheet, click in any empty cell and call the function in the usual way, passing it a small random number as the single argument:

A screenshot of the Google Sheets formula bar. It shows the function `=returnRowOfNumbers(5)` entered into the bar. The 'fx' icon is visible on the left side of the bar.

You should now see a row of numbers starting from 1 and counting up to the number passed in as an argument. This row started from the cell where you type in the function.

Instead of using an explicit number in the function call, you could also have referenced another cell containing a number and this will work perfectly fine too.

C5 *fx* =returnRowOfNumbers(A5)

	A	B	C
1			
2			
3			
4			
5	5		1
6			2
7			3
8			4
9			5

If you change the argument being passed to the function to a non-number type (for e.g. a string), an error message will be returned via the `throw` statement.

C5 *fx* =returnRowOfNumbers(A5)

	A	B	C	D	E
1					
2					
3					
4					
5	cat		#ERROR!		
6					
7					
8					
9					

Error
This function expects a number
! (line 20).

Copy the function `returnTableOfNumbers` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Back in Google Sheet, click in any empty cell and call the function in the usual way, passing it 4 small random numbers as the arguments, for e.g.

fx =returnTableOfNumbers(3,5,100,10)

	B	C	D	E	F
	100	101	102	103	104
	110	111	112	113	114
	120	121	122	123	124

Of course, you can also pass in the arguments via cell references as we did in the previous example.

1.3 Reading and writing data from and to a cell range in Google Sheet

A very common use case is to read values (either from a single cell, row, column, or a table) from a Google Sheet and then perform some custom processing on it, and then later write back some parts of the result to the same or different Sheet.

You can give the current untitled spreadsheet a name to save it.

Next, we will import an Excel spreadsheet with an existing table into a new Google Sheet. There are many way to accomplish this:

<https://spreadsheetpoint.com/convert-excel-to-google-sheets/>

In the current Google sheet, select File -> Import, and then select Upload.

Drag and drop the file `sample student marks.xlsx` from the `labcode` folder into this box.

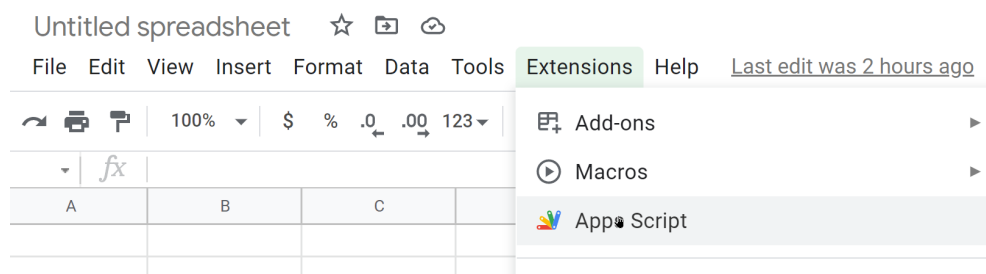
The file is uploaded, and then an Import File dialog box appears.

Select Import Data.

After the File imported successfully message appears, click on Open Now.

This should open a new browser tab holding a new Google Sheet with table data imported from the original Excel spreadsheet.

Close the previous Google Apps Script editor window, and open a new window from the new Google Sheet in the same way as demonstrated previously.

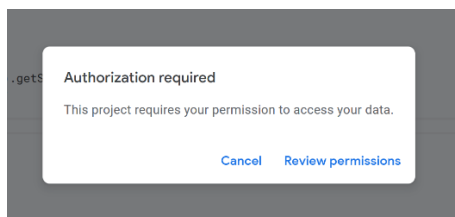


File to use: `gappscript-gsheet-basic.js`

Copy the function `showTableContent` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Run this function from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This simply prints out all the data from the 2-D array, row by row.

If you see any message related to authorization, such as that shown below:



Follow the sequence of steps outlined in:

<https://spreadsheet.dev/authorizing-an-apps-script>

Copy the function `findAverageMarkForStudent` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Run this function from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This prints out all the average mark for all the students.

You can verify that the average marks are calculated correctly by using the predefined `=AVERAGE()` function directly on the Google Sheet for all the rows. **MAKE SURE TO DELETE** the column after you are done, as the next function you run requires that the table be at its original size.

Copy the function `findAverageMarkForSubject` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Run this function from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This prints out all the average mark for all the subjects.

You can verify that the average marks are calculated correctly by using the predefined `=AVERAGE()` function directly on the Google Sheet for all the columns. **MAKE SURE TO DELETE** the row after you are done, as the next function you run requires that the table be at its original size.

Copy the function `createGradeTable` and `getGradeForMark` and paste it into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Run this function from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This creates a new table below the original table which shows the corresponding grades for each mark.

You can provide a name for the active Apps Script project that you currently have open, so that you can identify it the next time you open it from the same Google Sheet. That Google Sheet functions as the container to which the Apps Script project is bound to.

1.4 General approach to working with Google Apps Script

There are 4 general steps involved in creating a Google App Script program

- a) Determining the specific classes that you need to access (e.g. `SpreadsheetApp` to work with Google Sheets, `GmailApp` to work with Gmail, `DocumentApp` to work with Google Docs, etc) and the specific properties / methods within these objects that you need to access
- b) Using these properties / methods we obtain raw data from the various Google Apps (for e.g. a table from Google Sheet, or text from inside an email in Gmail or a document in Google Docs).
- c) We write standard JavaScript code using standard JavaScript constructs (if-else-if, loops, arrays, objects, functions, etc) to implement our custom business logic to process this raw data
- d) Using suitable properties / methods from the various classes, we place the processed data or results back inside another Google App and perform some action with it (for e.g. create a new email, place some text in it and send off the email).

The API Reference for Google AppS Script is available at:

<https://developers.google.com/apps-script/reference>

2 Try - Catch - Finally

File to use:

```
try-catch-basic.js  
throw-basic.js
```

The [try-catch block](#) is used to handle run time exceptions that may occur, so that the program can still continue running instead of being terminated immediately. The [try-catch-finally](#) block provides a block that will be executed regardless of whether exceptions occurred or not. We can always use the [throw](#) keyword to create our own custom Exception, to signal the detection of an issue that would count as an error in our code logic.

3 Promises

Files to use:

```
async-demo.js  
promises-anonymous.js  
promises-arrow.js
```

[Promises](#) are the main way we deal with asynchronous operations in JavaScript. [A Promise is an object](#) that is created by calling the Promise function and passing it a single function (called the executor) as its parameter. This function itself takes two parameters which are also functions: resolve and reject. When the `then` method is called on the Promise object, this method takes another two more callback functions as its parameters. These callbacks are passed to the resolve and reject parameters.

4 Working with JSON

File to use:

```
json-basic.js  
sample student marks.xlsx  
superhero.json
```

[JavaScript Object Notation \(JSON\)](#) is a widely used format of exchanging data between applications, particularly over the Web. Although JSON is based on JavaScript objects, it is designed to be used independently of JavaScript (or any other programming language). All major programming languages have libraries or functions that specifically deal with processing JSON data.

It is always a good idea to [validate the JSON](#) that you are working with to prevent unexpected errors when executing code to parse or process it.

There are several [main ways of working with JSON](#) in JavaScript directly. We can also [read from a JSON file](#) and convert it into an object and vice versa.

You can save an existing Excel spreadsheet into CSV form, and then [convert it into JSON form](#).

You can then read the JSON data and convert it into a JSON object.

As an exercise (using `sample student marks.xlsx`), you could try to replicate the functionality of common Excel formulas using a JavaScript program, for e.g.:

- Find the total / average mark for each student
- Find the average mark for each subject
- Find the 5 students with the highest / lowest marks for each subject
- Sort the students based on the mark for a given subject or average mark for all subjects

5 Using the Fetch API

File to use: `fetch-basic.mjs`

The Fetch API is already part of the JavaScript engine in modern browsers, so you can use it directly within a script referenced from a HTML document in a browser. For the case of a JavaScript program executed directly from Node.js, we first need to install the [node-fetch](#) package using NPM.

[NPM](#) is a package manager which is used for managing dependencies for a Node.js project. The dependencies are what the specific project needs to run correctly and usually consists of packages: which are a collection of JavaScript programs that perform functionality related to that project.

In a command prompt in the folder containing this file, type:

```
npm install node-fetch
```

The files for this package is installed in a newly-created subfolder `node_modules` within the current folder, and two configuration files `package.json` and `package-lock.json` are created as well. These files provide description about the project that is to be run in the current folder and any metadata related to it, such as the dependencies it needs to use.

Notice that the extension of this source code file is `mjs` instead of `js`, to indicate that it is a JavaScript module based on the [ECMAScript Module system](#). This is necessary for the Fetch library API to run correctly.

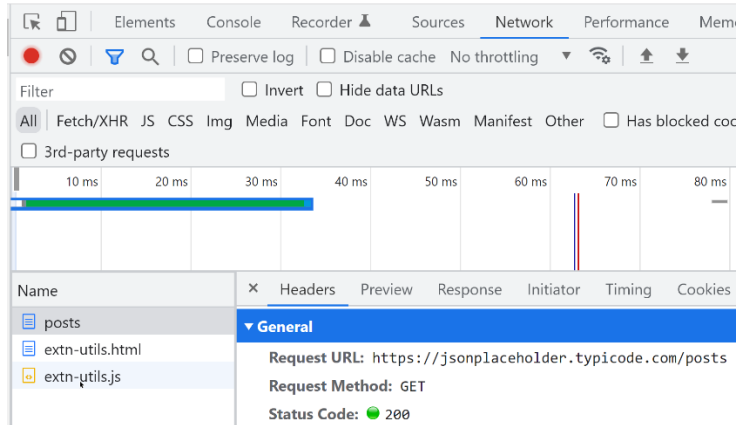
The [Fetch API](#) is a modern library that allows a JavaScript program to make HTTP requests to a web server. Typically the request will be made to a [REST API](#) using standard HTTP methods such as GET, POST, PUT, PATCH and DELETE. The most commonly used methods will be GET and POST. The Fetch API is based on promises.

We can use a [sample fake API service](#) to return JSON responses to test our JavaScript programs. You can check the [guide](#) on the various kind of resources you can retrieve using the Fetch API.

You can initially test out sending HTTP Get requests to these URLs by simply typing them into the address bar of the browsers, for e.g.

<https://jsonplaceholder.typicode.com/posts>
<https://jsonplaceholder.typicode.com/posts/1>
<https://jsonplaceholder.typicode.com/posts/5>

If you view the Network panel of the Chrome Developer Tools, you will be able to see the GET request in the Headers sent out as well the JSON sent back in the response message:



You can then test out the Fetch API by running the related JavaScript file in the usual manner:

```
node fetch-basic
```

6 Implementing a simple API server with Express.js

Files to use:

```
fetch-express.mjs
index.js
```

We can use Express.js, a minimalist web framework for Node.js to implement a very simple API server with can return a JSON response to a HTTP GET request issued from the browser or from the Fetch API in a JavaScript program.

First, select an empty directory anywhere on your machine (different from `labcode`) Open a new command prompt in this directory (leave your existing command prompt in `labcode` running). We will use the `npm init` command to create a `package.json` file first.

In the new command prompt, type:

```
npm init
```

and press Enter to accept all the defaults.

You will see the same two configuration files `package.json` and `package-lock.json` generated here as in the case of installing `node-fetch` in an earlier lab. These files provide description about the project that is to be run in the current folder and any metadata related to it, such as the dependencies it needs to use.

Next, we install Express.js with:

```
npm install express
```

Finally, copy over `index.js` from `labcode` into this folder and run it by typing:

```
node index.js
```

This starts up the web server implemented by the `index.js` program.

Open a new browser tab and navigate to this URL:

<http://localhost:3000/employees>

Notice that an array of 5 employee objects are returned.

You can retrieve specific employee objects by using this URL with the employee number appended to the end, for e.g.:

<http://localhost:3000/employees/1>

<http://localhost:3000/employees/2>

and so on.

In the command prompt in the `labcode` folder, there is a demo of using the Fetch API to retrieve the JSON responses from these URLs in a similar manner to the previous lab. You can run it with:

```
node fetch-express
```