

JavaScript

Intro for Beginners

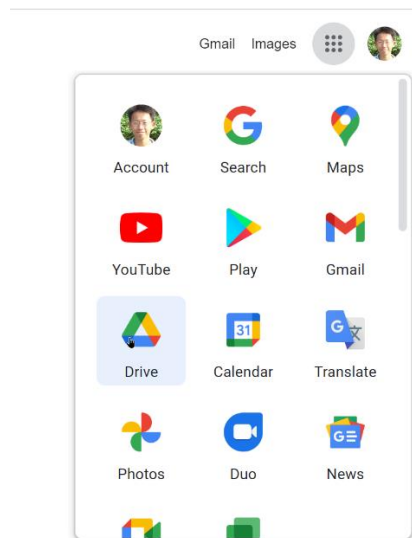
Lab 4

1	CREATING YOUR FIRST GOOGLE APPS SCRIPT.....	1
2	CALLING CUSTOM FUNCTIONS FROM GOOGLE SHEET	4
3	MULTIPLE WAYS OF ACCESSING A RANGE OF VALUES IN A SHEET.....	6
4	CUSTOM PROCESSING OF DATA RANGES	9
5	SENDING EMAIL WITH DATA FROM CUSTOMIZED PROCESSING	11
6	ADDING CUSTOM UI ELEMENTS TO GOOGLE SHEET	12
7	GENERAL APPROACH TO WORKING WITH GOOGLE APPS SCRIPT	13

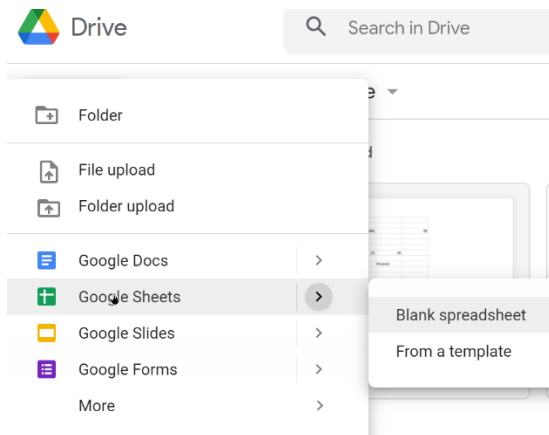
1 Creating your first Google Apps Script

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

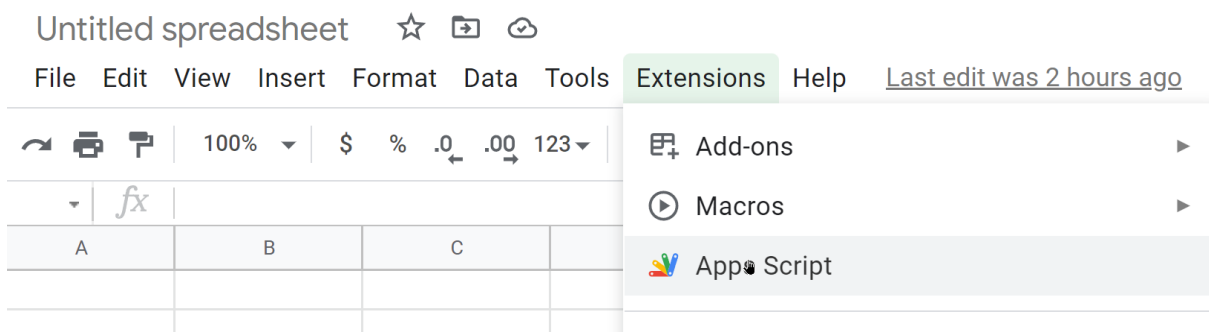
Make sure you are logged into your Gmail account and open a new browser tab in Chrome. Select the Drive icon from the drop down menu in the upper right hand corner.



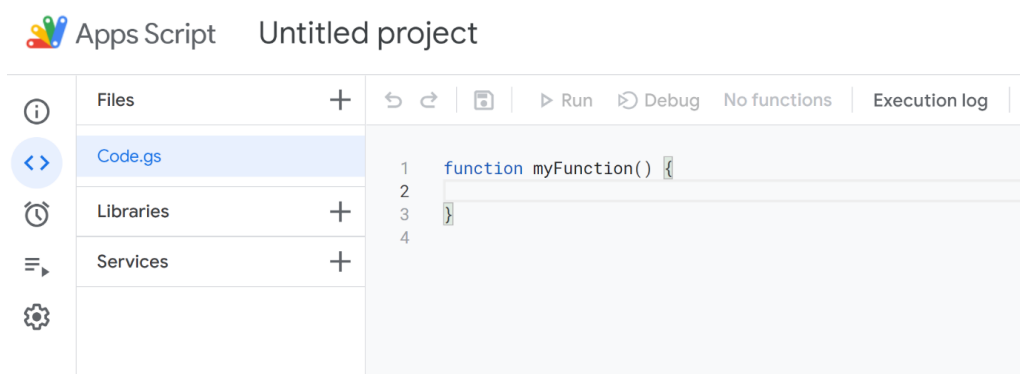
From the Drive menu panel on the left, select New -> Google Sheets to create a new Google Sheet.



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

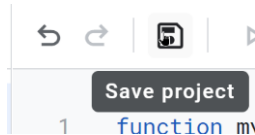


This opens up the Google Apps Script Editor view with a new untitled project and an empty function. This script is considered to be bound to this particular Google Sheet (a container-bound script) and once you save this sheet, you will be able to access all the changes (in the form of new functions) that you have last made into the Editor View.

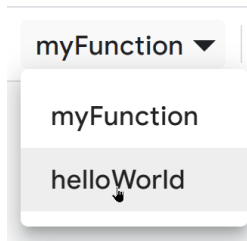


File to use: gappscript-intro.js

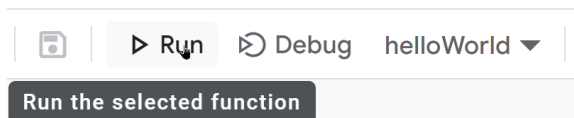
Copy the function `helloWorld` and paste it anywhere below the predefined empty function `myFunction`. Click Save Project (remember that you must always save your project before you run it).



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.

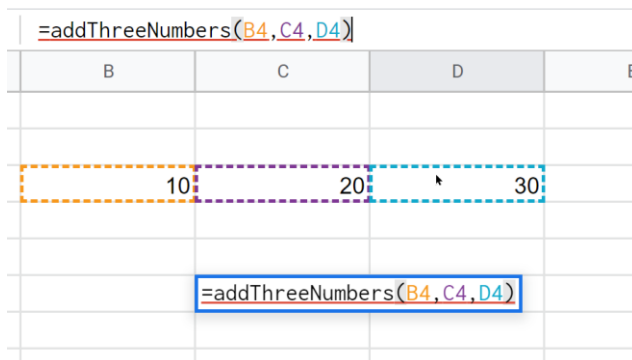
Copy the function `demoSampleStuff` and paste it the empty space at the bottom of the editor view and save the project (remember that you must always save your project before you run it). Run it and verify the output renders correctly in accordance to the logic of the code in it.

2 Calling custom functions from Google Sheet

File to use: gapps-script-intro.js

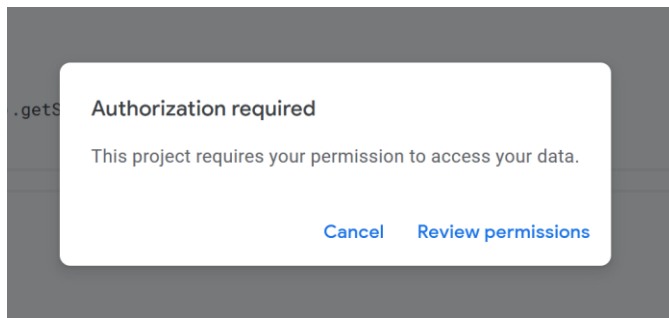
Copy the function `addThreeNumbers` and paste it the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

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.



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.

At any point of time, you can minimize the functions that are currently present in the AppScript Editor to make it easier to view these functions via the close / open symbol that appears to the left of each function.

```

5
6 > function helloWorld() {...
15 }
16
17
18 > function demoSampleStuff() {...
31 }
32
33 > function addThreeNumbers(firstNum, secondNum, thirdNum) {...
36 }
37

```

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 `returnColOfNumbers` 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:

=returnColOfNumbers(5)	
B	C
1	
2	
3	
4	
5	

You should now see a column of numbers appear starting from the cell where the function call is made, commencing from 1 and increasing by 1 up to the argument value.

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:

fx =returnRowOfNumbers(5)					
B	C	D	E	F	
1	2	3	4	5	

You should now see a row of numbers starting from 1 and increasing by 1 up to the argument value. 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.

A	B	C	D	E	F	G
6						
	1	2	3	4	5	6

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.

The screenshot shows an Excel spreadsheet with the following data:

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

The formula bar shows: `=returnRowOfNumbers(A5)`

An error message box is displayed next to cell C5, stating: **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.

	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.

Before going on to the next exercise, you can give the untitled spreadsheet that we were working with earlier a name to save it. You can later access it from your main Google Drive dashboard and the functions that you entered in the App Script Editor will still be present and bound to this spreadsheet.



My First Example   Saved to Drive

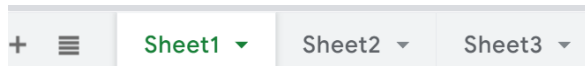
File Edit View Insert Format Data Tools

3 Multiple ways of accessing a range of values in a sheet

File to use: gappscript-demo-range.js

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.

Create a new empty Google Sheet. Add 3 new sheets to it using the + key at the bottom.



Populate all 3 sheets with the data shown here in the EXACT locations shown.

A1 fx 1

	A	B	C
1	1	11	21
2	2	12	22
3	3	13	23
4	4	14	24
5	5	15	25
6	6	16	26
7	7	17	27
8	8	18	28
9	9	19	29
10	10	20	30
11			
12			
13			

+ fx Sheet1 Sheet2 Sheet

A	B	C	D
31	41	51	
32	42	52	
33	43	53	
34	44	54	
35	45	55	
36	46	56	
37	47	57	
38	48	58	
39	49	59	
40	50	60	

+ fx Sheet1 Sheet2 Sheet3

A	B	C	D	E
	61	71	81	
	62	72	82	
	63	73	83	
	64	74	84	
	65	75	85	

+ fx Sheet1 Sheet2 Sheet3

Copy the function `accessRangeFirstSheet` 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 it to verify the different ways of selecting a sheet and a range of values in that sheet.

Copy the function `accessRangeSecondSheet` 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 it to verify the different ways of selecting a sheet and a range of values in that sheet.

Copy the function `accessRangeThirdSheet` 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).

Make sure Sheet3 is the active selected sheet and highlight a random range of numbers in this sheet.

B	C	D	
61	71	81	
62	72	82	
63	73	83	
64	74	84	
65	75	85	

Sheet1 Sheet2 Sheet3

Run it to verify the different ways of selecting a sheet and a range of values in that sheet.

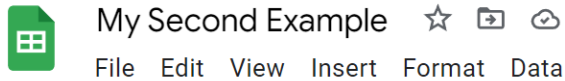
The various `getRange` methods can accept different ways of [referencing a range in Google Sheets](#), including A1 and [R1C1 notation](#).

This function uses the various classes related to the [Spreadsheet service](#) in order to access the content on the current spreadsheet that the script is bound to.

Some of the commonly used classes when working with accessing data ranges in a sheet are [SpreadsheetApp](#), [Spreadsheet](#), [Sheet](#) and [Range](#)

Create your own function to experiment with modifying the code provided to access random ranges of values in all 3 sheets.

Before going on to the next exercise, you can give the untitled spreadsheet that we were working with earlier a name to save it.



4 Custom processing of data ranges

File to use:

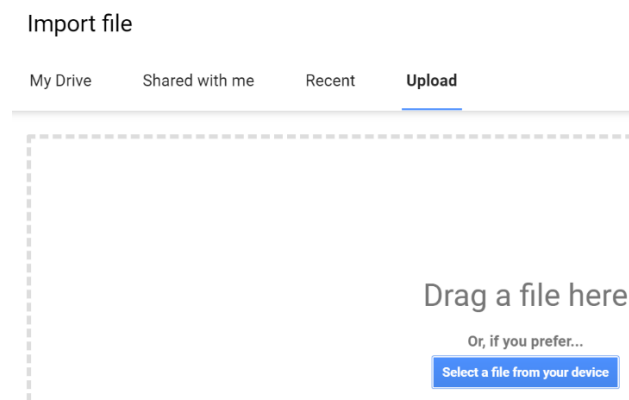
```
gappsript-custom-process.js  
sample student marks.xlsx
```

A common use case after accessing these range of values from a Google Sheet is to perform some custom processing on it, and then later write back some parts of the result to the same or different Sheet.

We will import an Excel spreadsheet with an existing table into a new Google Sheet. There are many ways to accomplish this:

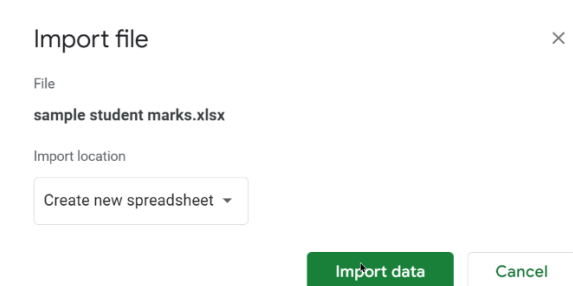
<https://spreadsheetpoint.com/convert-excel-to-google-sheets/>

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 with the option of Create new spreadsheet selected.



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

Import file

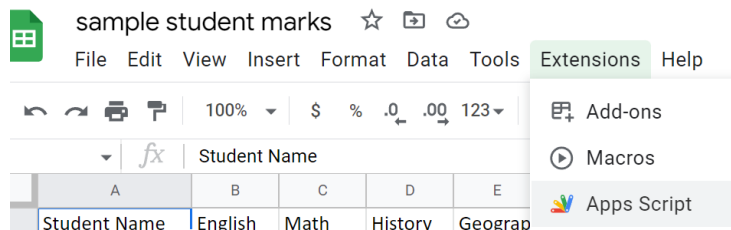
File

sample student marks.xlsx

File imported successfully. [Open now »](#)

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

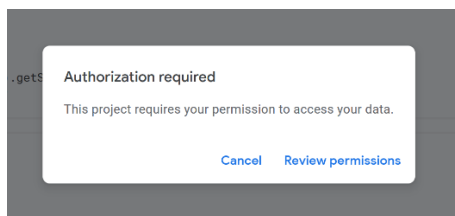
Close the previous Google Apps Script editor window and open a new Google Apps Script editor from the newly created Google Sheet in the same way as demonstrated previously.



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.

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>

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 them 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 the function `createGradeTable` 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. This is a form of custom processing using a Google App Script program which would be difficult to accomplish in a straight forward way using the existing functionality of Google Sheets.

5 Sending email with data from customized processing

File to use: `gapps-script-demo-email.js`

Another very common use for a Google App Script program is to transfer data between various Google Workspace applications (for e.g. between the contents of a Google Sheet into an email message sent via Gmail).

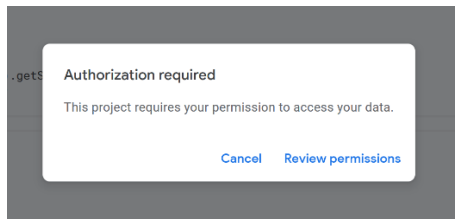
Add a random combination of valid and invalid email addresses in the column immediately after the end of the newly generated table from the previous section. The valid email addresses can be your own personal / work addresses or those of your teammates participating in this exercise.

15	Student Name	English	Math	History	Geography	
16	Student1	F	D	B	A	user1@gmail.com
17	Student2	C	B	A	F	user2@gmail.com
18	Student3	A	B	C	D	asdf
19	Student4	A	F	C	B	user3@gmail.com
20	Student5	B	C	F	D	user4@gmail.com
21	Student6	F	F	C	F	asdf
22	Student7	A	F	F	F	dddd@
23	Student8	C	F	B	A	eeeeee
24	Student9	A	B	C	D	user5@gmail.com
25	Student10	F	A	B	C	user6@gmail.com

Copy the functions `sendEmailBasicMessage` and `isValidEmail` and paste them 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 `sendEmailBasicMessage` from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This sends off a basic email message to the list of valid email addresses that you created earlier using the active Gmail account linked to the current Google account that the Google Sheet was created from. Verify that these emails are successfully received at their respective destinations.

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 `sendEmailCustomizedMessage` 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 extends on the previous function by creating a customized message for the student that summarizes the student's exam results. Verify again that these emails are successfully received at their respective destinations.

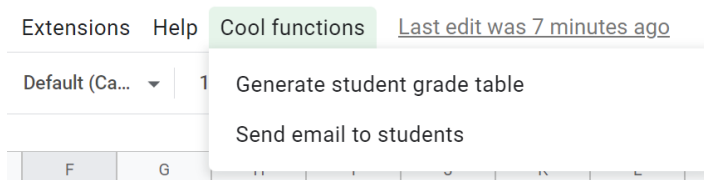
6 Adding custom UI elements to Google Sheet

File to use: `gappscript-demo-ui.js`

Once we have implemented a variety of customized functionality to process our Google Sheet data and transfer data to other Google Workspace apps, we may wish to simplify the process of triggering this functionality. Instead of having to explicitly choose the relevant function to run from the App Script editor window, we can create customized UI elements to the main menu (or other portions of Google Sheet) in order to trigger this customized functionality.

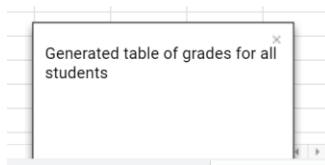
Copy the functions `customMenu`, `createGradeTableOption` and `sendEmailOption` and paste them into the empty space at the bottom of the editor view and save the project (remember that you must always save your project to persist the latest changes you have made to any functions defined there).

Run `customMenu` from the Google App Script editor, ensuring that you select this function from the drop-down menu from this place. This creates an additional customized item in the main menu bar of Google sheet, from which you can access the customized options:



Delete the table of student grades that was generated by the function `createGradeTable` in the earlier section.

Use the relevant customized option from the new menu item that we generated to perform this grade table generation again. Notice that a toast (message) also appears at the lower right hand corner of the view.



Enter in any random number of valid email addresses to the any of the 10 rows and then use the second customized option from the new menu item to send off a customized email message to those emails. Verify that another appropriate toast (message) appears at the lower right hand corner of the view and that the emails are successfully received at their respective destinations.

7 General approach to working with Google Apps Script

To summarize, 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 top level API Reference for Google Apps Script is available at:

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