Learn Google Apps Script Starter Guide

Getting Started with Google Apps Script

Google Apps Script is a coding language based on JavaScript that runs in the cloud. It allows you to interact with the Google Suite of products, and create customized functionality within those products. Google Apps Script, Google's server side Javascript-based cloud scripting platform for automating tasks across Google products and third-party services is available and can be accessed with an Google Account. You will need a Google Account in order to access the Google Suite. You can sign up for free at Google or if your company has an organizational account with Google

Workspace you can use that as well. There are some differences on permissions and who can access the documents between the organization accounts and the regular Google accounts.
Google Apps Script has many services that can be used, in addition to other Google Services that can be accessed with the code.
There is a reference guide located at
https://developers.google.com/apps-script/reference

To Get help there are resources online at
Questions and Code samples -
https://stackoverflow.com/questions/tagged/google-apps-script

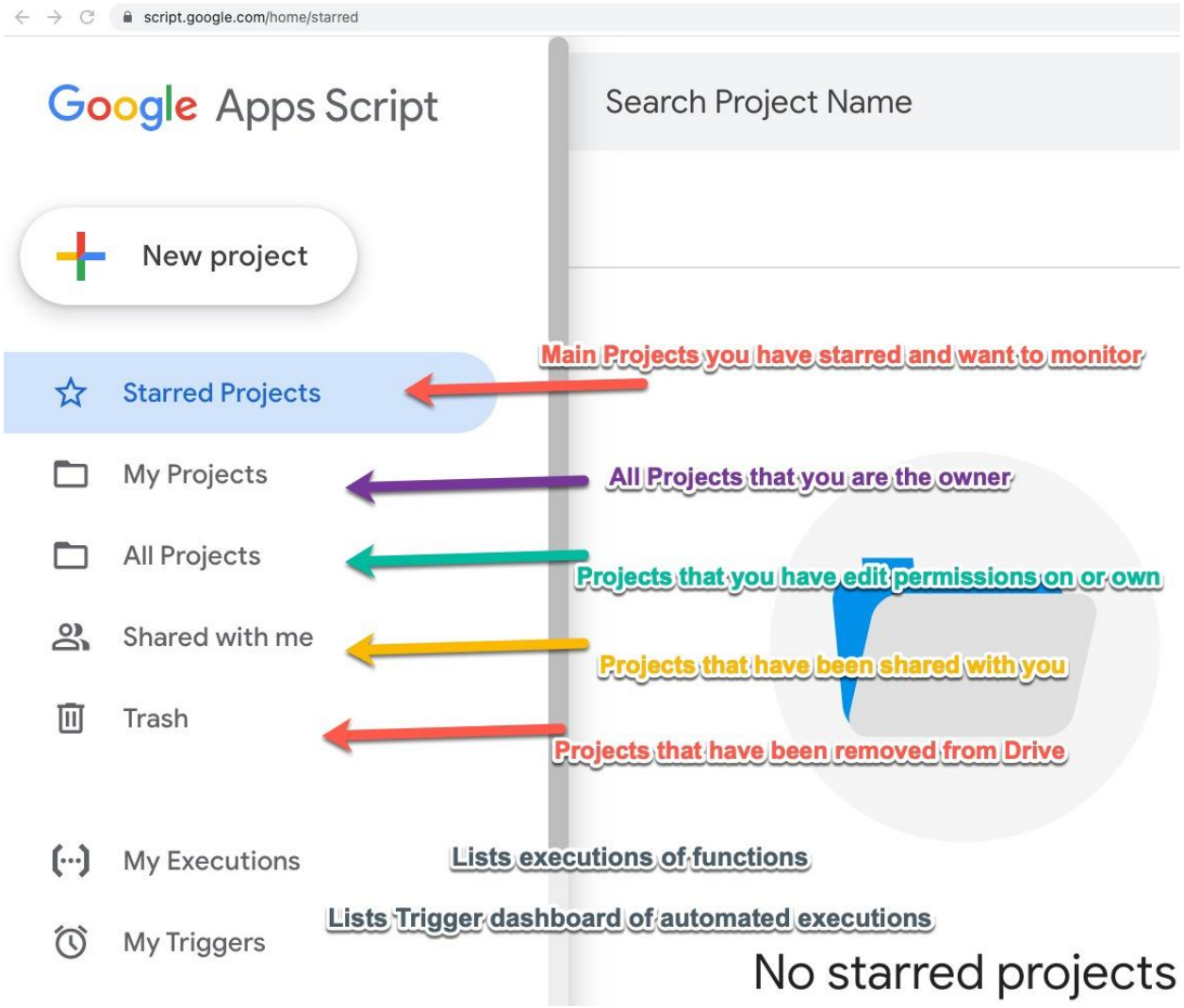
Open Google Apps Script from your Drive https://drive.google.com/drive/u/0/my-drive or create a new project from the Apps Script main page. https://script.google.com/home/

Learn more about apps script at https://developers.google.com/apps-script/

There is a lot that can be done with apps script, including **publishing web apps, creating custom functions** in Sheets, **adding menus** and dialogs to Docs, Sheets and Forms, **interacting with other Google Services** like Maps, Drive, Gmail, Calendar and much much more. You can also build and **publish addons to the Google Workspace Marketplace.**

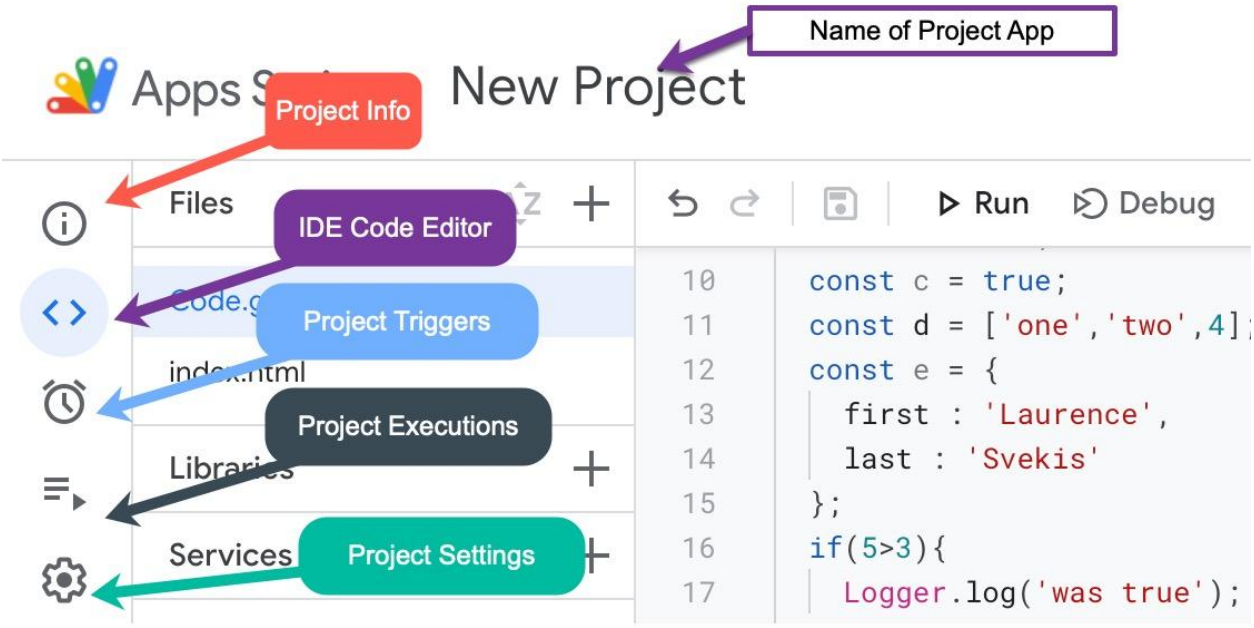
A prerequisite to being able to write App Script code is JavaScript. Apps Script code is written in modern JavaScript, in addition there are built in libraries that can be accessed within the Google Workspace products like Sheets, Docs, Gmail, Calendar, Drive and more. **There is nothing to install. You can write and edit the code directly in your Browser.**

Log into your Google Account and in the script dashboard create a new project. This will open the project in the Google Apps Script Editor. https://script.google.com/



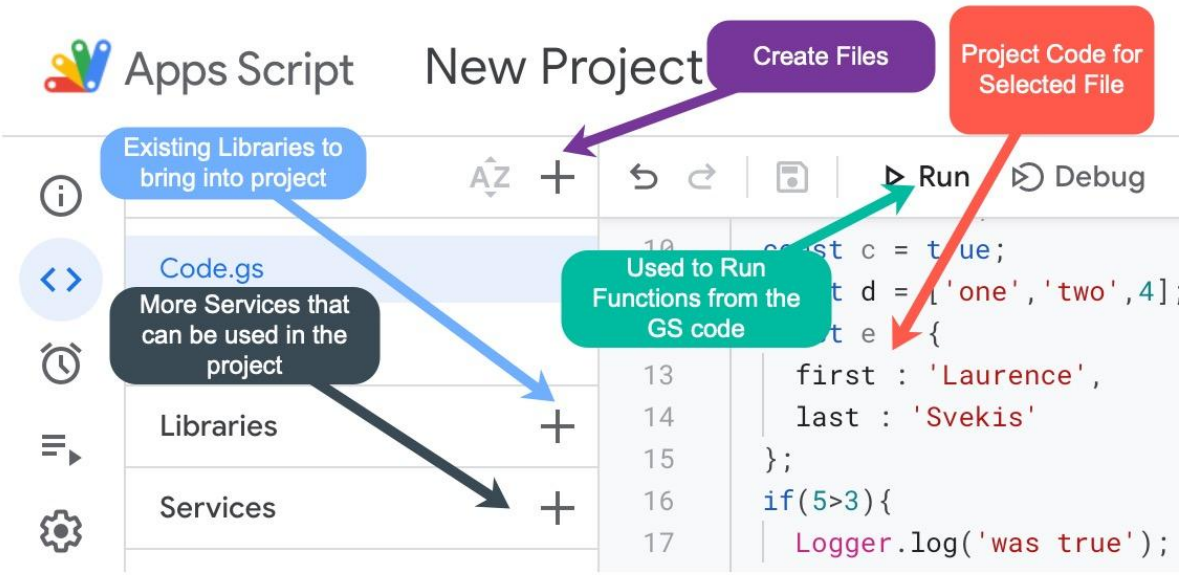
Give the project a meaningful name.

Open the IDE code editor screen to be able to write code.



By default the apps script editor will create a file called Code.gs and create a empty function in the editor called function myFunction(){}

Add a line of code within the function Logger.log('hello'); and using the Run button select the function and execute the code.



The Logger is a built-in function in apps script that allows you to send data to be displayed in the execution log. The execution log will display once the function is run if the Logger function is called. This log is lightweight and streams in real time, but

persists only for a short time and can be used for error checking, debugging and retrieving information as the code runs.

In the code editor you can write regular JavaScript code, as Apps Script and JavaScript code runs within a runtime or runtime environment containing the JavaScript engine that parses and executes script code.

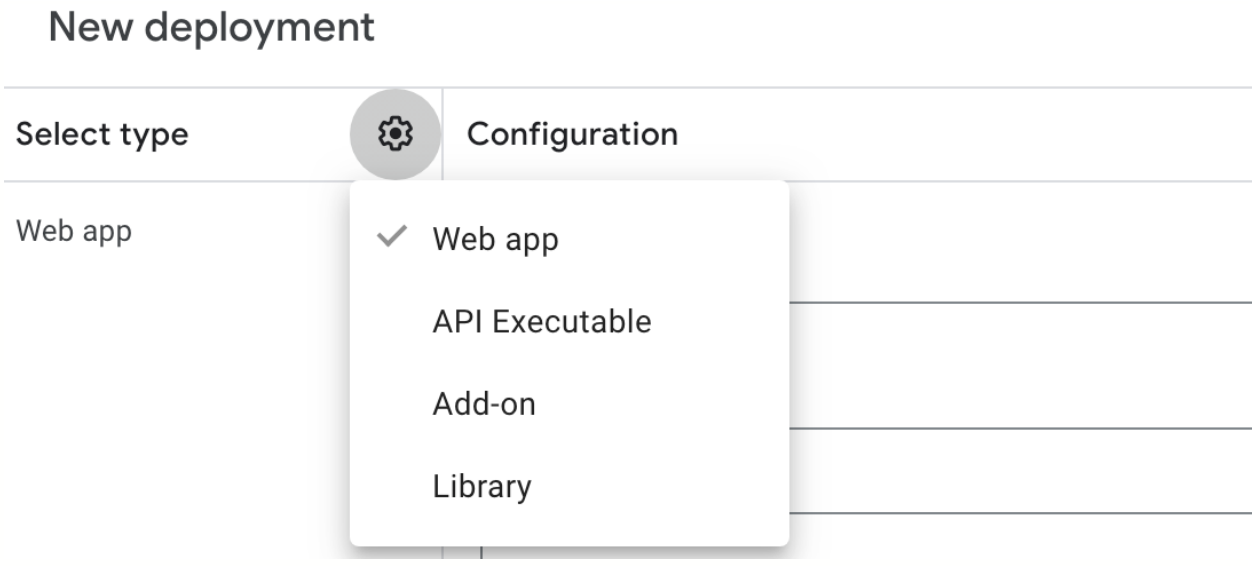
Within your code editor add some JavaScript code, run the function. In order to run the code it should be contained within a function that can be selected from the drop down menu.

```
function testScript1() {
 Logger.log('Hello World');
 for(let x=0;x<10;x++){
   Logger.log(x);
 }
}
function testScript2() {
 const a = 'string';
 const b = 100;
 const c = true;
 const d = ['one','two',4];
 const e = {
   first : 'Laurence',
   last : 'Svekis'
 };
 if(5>3){
   Logger.log('was true');
 }
```

```
 d.push('three');
 d.forEach(ele=>{
   Logger.log(ele);
 })
 Logger.log('Hello World');
}
```

Deploying a web app from a standalone application.

You can create a web app with Google Apps Script and publish the script as a web app. Both standalone scripts and bound scripts can be used to deploy web app, although best practice is to use the standalone script to do so.
The web app must contain either doGet(e) or doPost(e) function and return an output value in order to be able to respond.
With the Deploy button you can also deploy the app as either a web app, library, API executable or Addon. Once you select the deployment type fill out the configuration details.



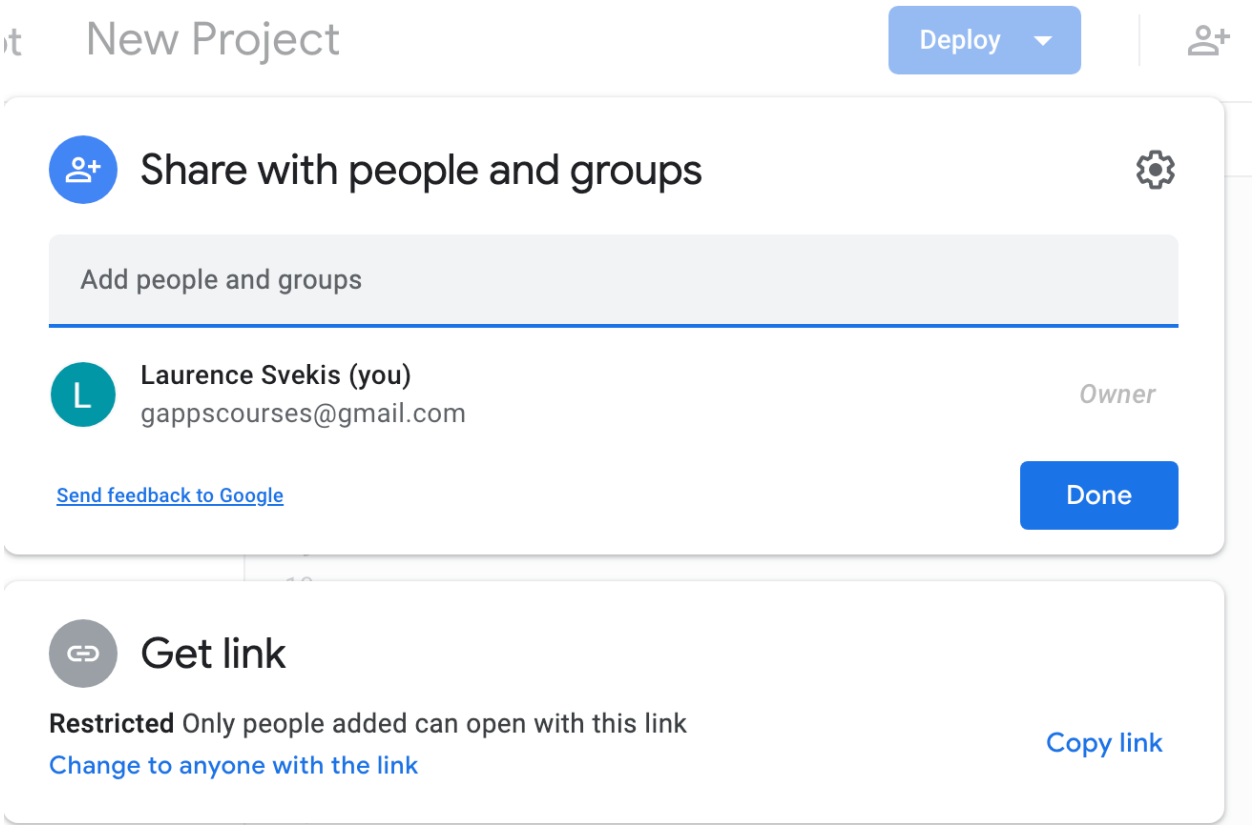
You can manage existing deployments, to either update or create new versions. Also once you deploy you will be able to see the deployments in the Test deployments tab.

Using the test deployment allows you to update your script and see the changes right away in the dev test deployment URL, which will end with dev. To update and make changes to your live version which will have a URL ending with exec, you need to redeploy once the changes are made.

Create your own web app, using the doGet() return back a custom message with HTML to be output into your web app.

```
function doGet(e){
 const html = `Hello Laurence Svekis 2`;
 return HtmlService.createHtmlOutput(html);
}
```

Share options are also available, just like other workspace apps, Apps Script can also be shared and collaborated on. Edit permissions will allow others to edit the code, view will only allow viewing of the code.



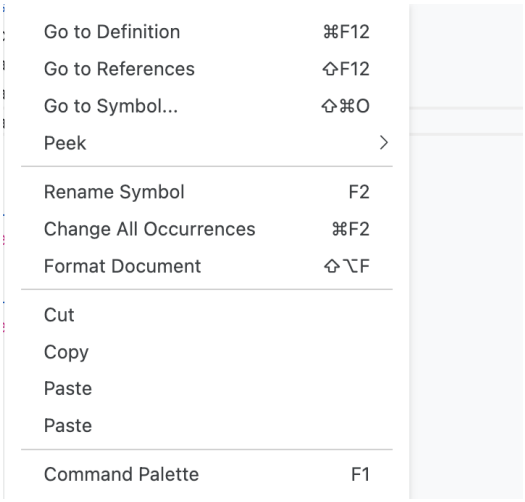
How to create a Bound Script

Bound scripts are available in Google Docs, Slides, or Forms. They provide some additional functions for interacting with the container file using Apps Script. The editor and script will be the same and code will run the same way.

Can be opened from the container file or from the script dashboard. Bound scripts will have a different icon which will include the parent container icon with a small script icon on top.

Bound script can add custom functions into a spreadsheet, run special triggers and function to interact with the container file.

To open the editor menu, right click or Control+Click within the editor. This will open the editor menu.



Bound scripts can add UI menus to a container file, within sheets using the UI object add a custom menu.

1. Using the onOpen() run a block of code as the spreadsheet opens.
2. Select the Ui object and add several items to the menu. Create the functions, use strings to represent both label and function names.
3. Create a couple of functions that will use the UI and open alerts with function messages.

```
function onOpen() {
 const ui = SpreadsheetApp.getUi();
 ui.createMenu('adv')
   .addItem('test1', 'test1')
   .addItem('test2', 'test2')
   .addToUi();
}

function test1() {
 SpreadsheetApp.getUi().alert('test1');
}
function test2() {
 SpreadsheetApp.getUi().alert('test2');
}
```

Learn more about Google Apps Script Services

Google Apps Script services provide ways for your script to access data on Google and external systems. The services are built in and don't require any import to be able to use them within your apps script code. Google Services allow you to use the data of your Google Workspace apps. There are also utility services that are not connect to a particular Google product, but they are useful to be able to do some amazing things within your code.

Within the resources you can find out more about the various classes for the apps, with code snippets.

// The code below will write "Esta es una prueba" to the log.

var spanish = LanguageApp.translate('This is a test', 'en', 'es');

Logger.log(spanish);

To run code, add it to your apps script, and wrap the code within a function. This can then be run with the Run button. Try out the LanguageApp service below.

```
function test3(){
 const val  = 'Hello World';
 const spanish = LanguageApp.translate(val, 'en', 'es');
 Logger.log(spanish);
}
```