

Learn Google Apps Script

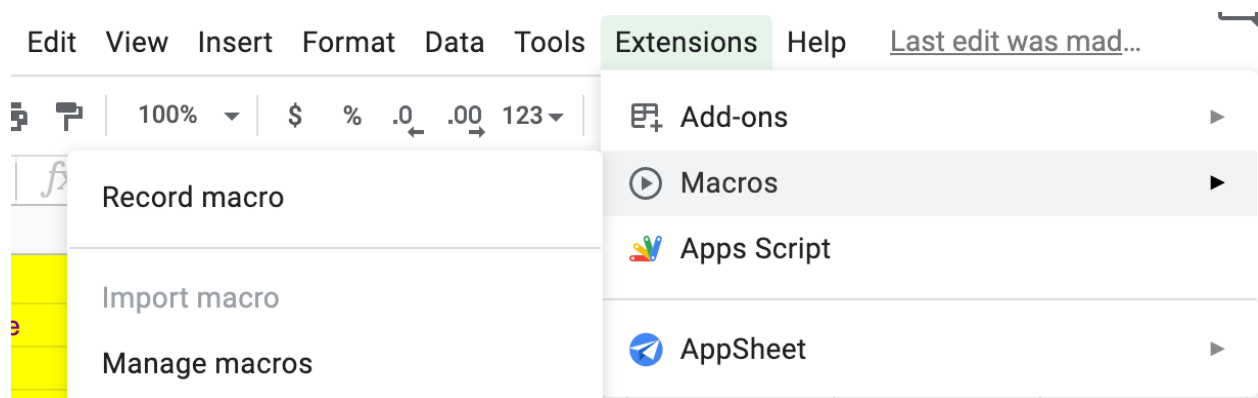


Introduction to Google Apps Script	1
Creating PDFs and files in Drive	12
Create a PDF Email from Sheet data	14
UI Menu Maker	16
Custom Functions in Sheets	21
Web Apps with Apps Script	24
Do more with Web Apps	27
doPost and doGet as an endpoint for web applications.	33
ClientSide to ServerSide WebApp	39
Google Apps Script Triggers for Automation	43

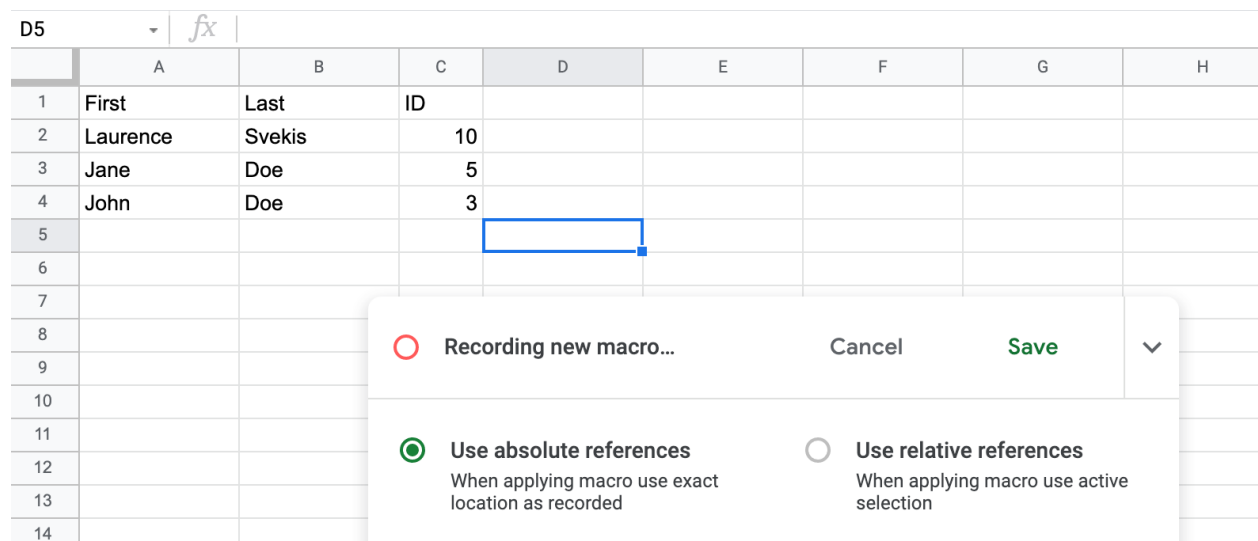
Introduction to Google Apps Script

Macros within Google Sheets can be used to write Apps Script Code. Create a new spreadsheet and add some data into the sheet. <https://docs.google.com/spreadsheets/>
In the top menu under extensions select Macro.

Google Apps Script is created using standard JavaScript Syntax.



Apply some changes and update the text color of some cells within your Sheet. Then Select Save in the Macro recorder to end the recording of the macro.



Give the macro you created a name and press save.

Save new macro

Name

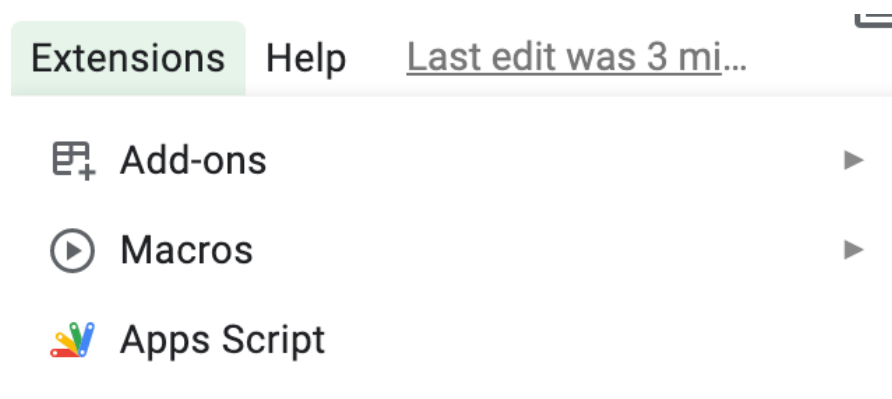
My Macro

Shortcut (optional)

⌘ + Option + Shift + Number

Discard Save

Select the Apps Script menu item under the Extensions tab to open the Apps Script editor.



This will open the editor, where you can see the created Apps Script code. Below code selects the range A1:A4 and activates it. `“spreadsheet.getRange('A1:A4').activate();“` Next using the active range it applies the font color of blue to the cell contents. `“Spreadsheet.getActiveRangeList().setFontColor('blue');“`

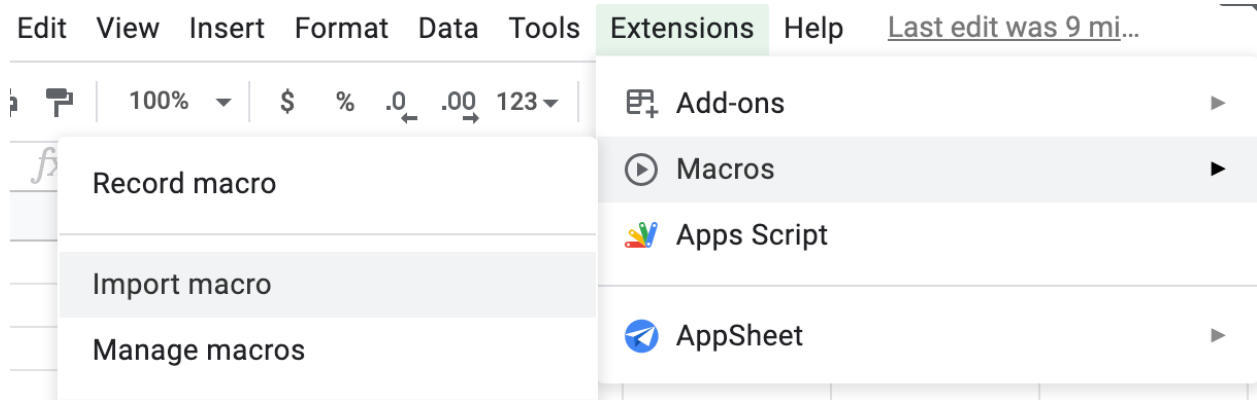
```
function MyMacro() {
  var spreadsheet = SpreadsheetApp.getActive();
  spreadsheet.getRange('A1:A4').activate();
  spreadsheet.getActiveRangeList().setFontColor('blue');
};
```

You can now edit and update the Apps Script code. This will still run under the macro name every time that macro is selected.

Next copy and paste the macro code. Create a new function with a new name.

```
function makePurple() {
  var spreadsheet = SpreadsheetApp.getActive();
  spreadsheet.getRange('A1:C4').activate();
  spreadsheet.getActiveRangeList().setFontColor('purple').setBackground(
    'yellow');
};
```

Go back into the Google Sheet, select Extensions then macro in the menu. There should be an option to import the macro. Select that and then import the new function as a macro into Google Sheets. Select macro under the menu again, you should see a new macro with the same name as your function, which you can now click and run.



Import



Added functions will appear in the menu under Extensions > Macros

makePurpleText

Project: Recorded Macros (tester1) File: macros





Edit Script





Google Apps Script allows you to create custom blocks of code that do stuff, and there is a lot of stuff you can do with it.

Container-bound Scripts - the Google file the Apps Script is attached to is known as the container. Bound scripts behave just like Standalone scripts but they have several special functions, they do not show up in drive and they cannot be detached from their container files. Script can be bound to Sheets, Docs, Slides, or Forms. To reopen the script in the future you can open the container file and select the script editor, or you can go directly to the <https://script.google.com/> Apps Script home page, and select the project from there. For bound scripts it will show the container file icon, with a small Apps Script logo on it indicating that it is a bound script.

My Projects

Project ↑	Owner
 tester1	Me
 Recorded Macros (Test New (MAIN)) 	Me
 Sheets Test Code	Me

Standalone scripts are indicated with the Apps Script logo as in the example below.

 Example Code	Me
 EMAIL TEMPLATEgmailApp Tester	Me

You can remove permissions at <https://myaccount.google.com/permissions>

Send emails to users from a Google Spreadsheet using Google Apps Script.

1. Get the spreadsheet object - because this is a bound script you can use `SpreadsheetApp.getActiveSpreadsheet()`.
2. Select the Sheet object `getActiveSheet()` which will return the sheet that is currently selected in the spreadsheet.
3. Select the range for the data that you want to use, `getDataRange()` will select the entire contents of all the data in the sheet.
4. Once you have the range of data cells you can now get the values into an array, each row will be a separate nested array within an array for the entire sheet.
`getValues()`

5. To remove the first row of data which is typically used for headings, slice(1) the values starting at index 1.
6. Iterate through all the items in the main array, each item is a row of data from the sheet. `data.forEach((row)=>{})`
7. Create the HTML message as a variable, the email as a variable, and the subject as a variable.
8. Use MailApp to send the email using `sendEmail` method. *You will need to accept permissions for the app.

```
MailApp.sendEmail({
  to : email,
  subject : subject,
  htmlBody : message
});
```

```
function sheetDataTest() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getActiveSheet();
  //Logger.log(sheet.getSheetName());
  const range = sheet.getDataRange();
  //Logger.log(range.getValues());
  const res = range.getValues();
  const data = res.slice(1);
  /*for(let i=0;i<data.length;i++){
    Logger.log(data[i]);
  }*/
  data.forEach((row)=>{
    //Logger.log(row[3]);
```

```

    const message = `Hi, ${row[0]} ${row[1]} <p> Welcome to the
site.</p><hr>
    <div> Your id is <br><h1>${row[2]}</h1></div>`;
    const email = row[3];
    const subject = `Test email ID ${row[2]}`;
    //Logger.log(message);
    //MailApp.sendEmail(email, subject, message);
    MailApp.sendEmail({
        to : email,
        subject : subject,
        htmlBody : message
    });
})
}

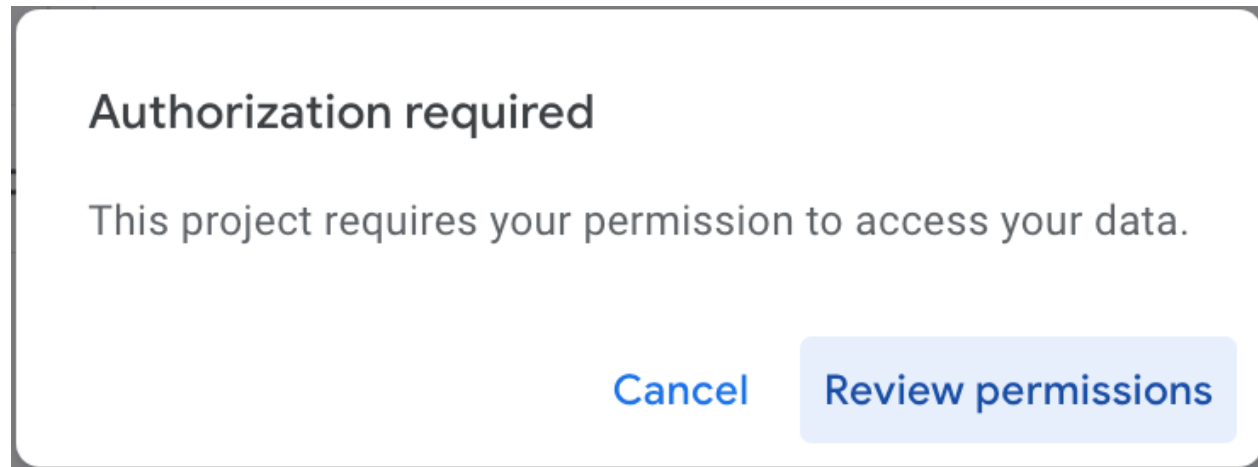
```

Whenever your code needs to run methods that need permissions, you will see the permissions popup screen. This ensures that your account is accepting to have the Apps Script run using your account. You will need to accept permissions to run Apps Script.

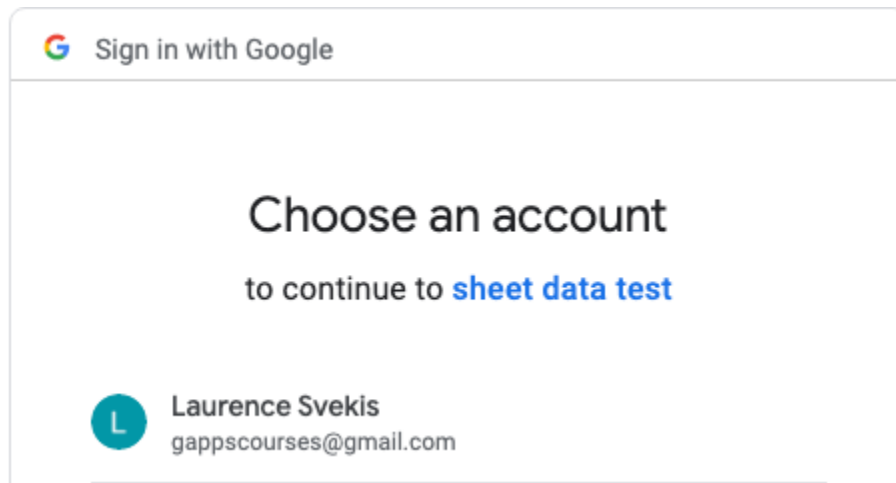
To RUN a function select the Run button from the menu, as well ensure you have selected the correct function by its name that you want to run.

▶ Run ▶ Debug sheetDataTest ▼

Select the Review permissions link in the Authorization screen popup.



Select the Google account to use to run the code.



Select the Advanced link to continue



Google hasn't verified this app

The app is requesting access to sensitive info in your Google Account. Until the developer (gappscourses@gmail.com) verifies this app with Google, you shouldn't use it.

Advanced

BACK TO SAFETY

Go to the app by its name



Google hasn't verified this app

The app is requesting access to sensitive info in your Google Account. Until the developer (gappscourses@gmail.com) verifies this app with Google, you shouldn't use it.

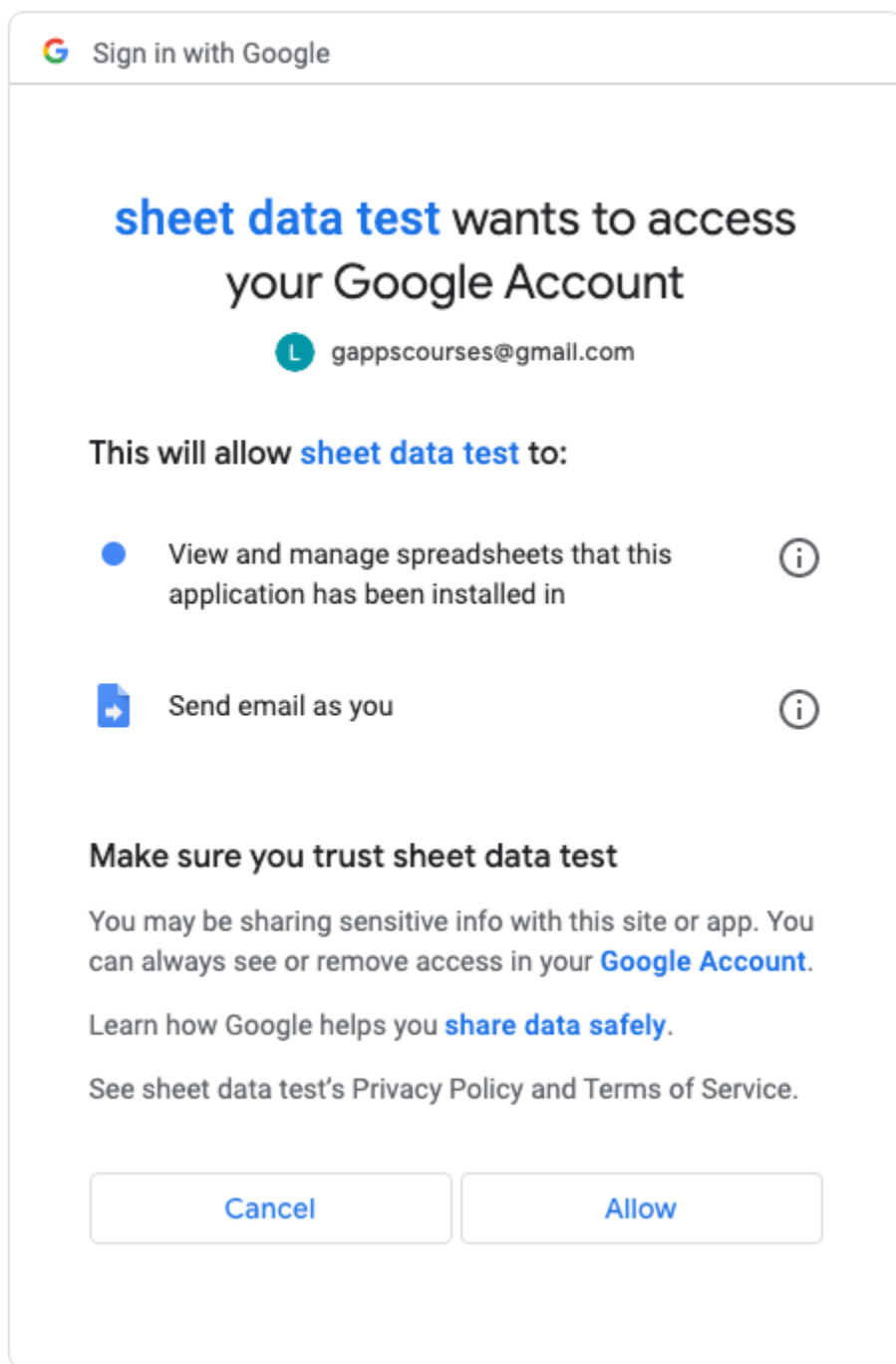
Hide Advanced

BACK TO SAFETY

Continue only if you understand the risks and trust the developer (gappscourses@gmail.com).

[Go to sheet data test \(unsafe\)](#)

Review the access request window, review permissions you are granting. You can always remove the app permissions at <https://myaccount.google.com/permissions>



Selecting allow will allow the app to use your Google Account to run the code in the app.

Creating PDFs and files in Drive

Create a script that will get data from a sheet, create content to add within a Doc. Make the Doc, update the sheet data with the doc details, and email out to the user from the sheet the doc location.

1. Select a Spreadsheet sheet by Name - `getSheetByName('data')`
2. Get the data from the sheet
3. Loop through the sheet data, invoke a function `makerDoc()` sending the row array values and the index value.
4. Create a new function `makerDoc()`, set a value for the row using the index from the array. The Array is zero based and the first row contains headings so we add 2 to the value to get the actual row value.
5. Create variables for the message within the doc, and the name of the doc which can be generated from data sent into the function.
6. Create a doc and select the doc body object. `DocumentApp.create(docName); doc.getBody();`
7. Using the body append a paragraph with the message contents.
`body.appendParagraph(bodyData);`
8. Get the newly created doc's url. `doc.getUrl();`
9. Create values to send an email, including email address, subject and message body. Send the email with `MailApp`.
`MailApp.sendEmail(email,subject,emailBody);`
10. Return the row value and the message for the spreadsheet cell back.
11. Use the response object from `makerDoc()` to select a range in the sheet, then update and set a value from the response object message.
`sheet.getRange(val.row,5).setValue(val.message);`

```
function myData(){
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheetByName('data');
```

```

const rows = sheet.getDataRange().getValues();
const data = rows.slice(1);
data.forEach((row, ind)=>{
    const val = makerDoc(row, ind);
    const updateRange = sheet.getRange(val.row, 5);
    updateRange.setValue(val.message);

})
//Logger.log(data);
}

function makerDoc(data, ind){
    //Logger.log(data);
    const rowVal = ind + 2;
    let message = `${data[0]} ${data[1]}`;
    const docName = `${data[0]} ${data[1]} ${rowVal}`;
    const doc = DocumentApp.create(docName);
    const bodyData = `${docName} Hello and Welcome. your id is
    ${data[2]}`;
    const body = doc.getBody();
    body.appendParagraph(bodyData);
    const url = doc.getUrl();
    message += ` Your doc is located at ${url}`;
    body.appendParagraph(message);
    const email = data[3];
    const subject = `${docName} created`;
    const emailBody = `Your doc is created at ${url}`;

```

```

MailApp.sendEmail(email, subject, emailBody);
//Logger.log(docName);
return {row:rowVal,message:message};
}

```

Laurence Svekis 2

3:06 PM

Jane Doe 3

3:06 PM

John Doe 4

3:06 PM

☐ ☆ me

John Doe 4 created - Your doc is created at https://docs.google.com/open?id=1qe0_00ouD28SGh3-uZ-qEuCARKhic6K0U-Hzn3przDw

John Doe 4

☐ ☆ me

Jane Doe 3 created - Your doc is created at <https://docs.google.com/open?id=1xHzLZIOXnYy2HvNiMdHP36Bq0ZIJKH5L1NPFD43VGLg>

Jane Doe 3

☐ ☆ me

Laurence Svekis 2 created - Your doc is created at https://docs.google.com/open?id=1dt8DG-GOWUgcoJ_9hQDdl-IYzF8Y_dhfNVma7oEnOU

Laurence Svekli...

fx

A	B	C	D	E	F	G
First	Last	ID	Email	Doc		
Laurence	Svekis	10	gappscourses+1@gmail.com	Laurence Svekis	Your doc is located at https://docs.google.com/open?id=1dt8DG-GOWUgcoJ_9hQDdl-IYzF8Y_dhfNVma7oEnOU	
Jane	Doe	5	gappscourses+2@gmail.com	Jane Doe	Your doc is located at https://docs.google.com/open?id=1xHzLZIOXnYy2HvNiMdHP36Bq0ZIJKH5L1NPFD43VGLg	
John	Doe	3	gappscourses+3@gmail.com	John Doe	Your doc is located at https://docs.google.com/open?id=1qe0_00ouD28SGh3-uZ-qEuCARKhic6K0U-Hzn3przDw	

Create a PDF Email from Sheet data

Generate a PDF document on the fly and send that document as an attachment to an email with all the data coming from a spreadsheet. Standalone script connecting to a spreadsheet for data and updating the selected spreadsheet.

1. Get the ID of your spreadsheet that contains the data.

<https://docs.google.com/spreadsheets/d/1yXiEb6EDJIRyEAskPqiYyNh52pCcVLOJ2t9dzJKvneU/edit#gid=0>

2. Use the id and open the sheet by the id. SpreadsheetApp.openById(id);

3. Select the sheet in the spreadsheet with the data you want, get the values without the headings as an array of rows.
4. Loop through the rows of data, using the data from the Spreadsheet to create some HTML code.
5. Use the Apps Script Utilities to create a blob of the html content.
Utilities.newBlob(html,MimeType.HTML);
6. Set a name for the blob. blob.setName(`\${row[0]} \${row[1]}.pdf`);
7. Get the email address, subject to send the PDF to.
8. Using MailApp send an email to the user, with html for the body and subject.
Using the attachments property in the email add the blob of PDF.
attachments:[blob.getAs(MimeType.PDF)]
9. Update the spreadsheet row data with a value that the PDF was sent.
10. Check the inbox for the PDF and the email.

```
function pdfMaker() {
  const id = '1yXiEb6EDJlRyEAskPqiyyNh52pCcVL0J2t9dzJKvneU';
  const ss = SpreadsheetApp.openById(id);
  const sheet = ss.getSheetByName('data');
  const data = sheet.getDataRange().getValues().slice(1);
  data.forEach((row, ind)=>{
    let html = `

# ${row[0]} ${row[1]}</h1>`; html += ` Welcome your id is ${row[2]}</div>`; html += ` Thank You</div>`; const blob = Utilities.newBlob(html,MimeType.HTML); blob.setName(`${row[0]} ${row[1]}.pdf`); const email = row[3]; const subject = `New PDF ${row[0]} ${row[1]}`; MailApp.sendEmail({ to:email,


```

```

        subject:subject,
        htmlBody:html,
        attachments:[blob.getAs(MimeType.PDF)],
    });
    sheet.getRange(ind+2,5).setValue('sent');
    Logger.log(blob);
  })
}

```

UI Menu Maker

Bound scripts can use a trigger like the special `onOpen()` function, which runs automatically whenever a file is opened by a user who has edit access. Using this with the ui menu maker will automatically add the menu item to the spreadsheet and allow anyone who has edit permissions access the functions.

How to add a custom UI menu to your spreadsheet.

Using the `onOpen()` method will add the UI menu buttons to the spreadsheet menu when the application opens. If you need additional functions to run within the sheet when it opens you can add all the functions within this one `onOpen()`.

```

function onOpen() {
  const ui = SpreadsheetApp.getUi();
  ui.createMenu('Adv')
    .addItem('Test 1', 'test1')
    .addItem('Test 2', 'test2')
    .addItem('Test 3', 'test3')
    .addToUi();
}

```



```
}
```

How to send a custom message to the UI alert.

Create a function that will send a message string to be output into the spreadsheet alert window. This can now be used to create a popup alert with a custom message within your script.

```
function outputMessage(val){
  SpreadsheetApp.getUi().alert(val);
}
```

How to select the values of the selection range cells.

Create a function that will get the selected values and output them into the alert menu content. You can use the `getActiveRange()` method to select the range of values to be used. Use `getValues()` to return the array of cell contents in a nested array of rows nested into a main array.

```
function test1(){
  const ss = SpreadsheetApp.getActiveSheet();
  const data = ss.getActiveRange().getValues();
  let mes = '';
  data.forEach(ele =>{
    ele.sort();
    let output = ele.join('|');
    mes += output + '\n';
  })
  //Logger.log(mes);
  outputMessage(mes);
}
```

How to search the selected range cells for a value and then determine which cell matched the value and update the found cell properties.

Create values in your spreadsheet that you want to check for, if found that the cell array row item is equal to this value you can get the cell range and update the cell properties.

1. Within the function select the values of the active range selection when the button is pressed and the function runs.
2. Loop through the contents of the data, as you loop through the rows using `indexOf` check to see if the value of any of the items in the row array is equal to the value we are searching for. In the example I use the value "Laurence" The cell value must be equal to the value we are looking for, if there are any extra characters it will not return an index value within the row array.
3. If the value is found, the `indexOf` will return the index value, if no match is found the response will be -1. We can use this in a condition to check if the returned value is -1 or not.
4. Using the `getRow()` and `getColumn()` methods from the selection range, you can find the starting cell for the selection. The `ind` of the row array, and the `indexOf` of the matching column value, can then be used to add to the starting cell coordinates and calculate the current cell coordinates in the spreadsheet.
5. Use the calculated values of the row and column for the match, and get it as a range. Once you have the range you can then update the cell properties like `setBackground()`.
6. Update the background color to yellow of the found cell.

```
function test2(){
  const ss = SpreadsheetApp.getActiveSheet();
  const data = ss.getActiveRange().getValues();
  data.forEach((row, ind) =>{
    const checker = row.indexOf('Laurence');
    if(checker !== -1 ){
```

```

    Logger.log(checker+1);
    Logger.log(ind+1);
    const range = ss.getActiveRange();
    const x = range.getRow() + ind;
    const y = range.getColumn() + checker;
    const cell = ss.getRange(x,y);
    cell.setBackground('yellow');
    //outputMessage(`${x} x ${y}`);
  }
})
}

```

How to update the selection cell values to new values.

Getting the selected range value, the dimensions of the array will need to match to the new value array in order to be able to update the range with the new values.

The array method `map()` allows us to return updated values of the array into a response that can then be used to create a new array.

1. Select the active data cell values.
2. Create a holding array that will be used to add the updated value to, as well as a starting value to a counter that will be added to the cell values.
3. Loop the data values for each row. Using `map` update and returning a new array for the row values. Add this newly created array for the row values into the holder array that will then be used to update all the cell content.
4. Within the `map` method, check if the cell value contains a dot. If it does then break the content at the dot removing the values up to the dot. This is needed so that once the cell has the value and the dot placed, we can remove the previous value and add a new one to the cell contents. First determine if the cell value has a dot already, this can be done using the `indexOf` which will return the dot index value or -1 if no dot is found.

5. Using trim() method for strings you can remove surrounding whitespace from the cell content
6. Convert the cell content to a string using the toString() method. They need to be strings in order to use the string methods in Apps Script on the values.
7. Increment the counter value by one for each new cell that will be written. Add the counter value with a dot separating it from the existing content.
8. Once the holder array is created, it can now be used to update all the selected cell values using setValues().
9. You can also clear any existing formatting using the clearFormat() method.

```
function test3(){
  const ss = SpreadsheetApp.getActiveSheet();
  const range = ss.getActiveRange();
  const data = range.getValues();
  const holder = [];
  let counter = 0;
  data.forEach((row)=>{
    const upArr = row.map((val)=>{
      const dotInd = val.toString().indexOf('.');
      //Logger.log(dotInd);
      let output = val.toString().trim();
      if(dotInd !== -1){
        output = val.toString().slice(dotInd+1).trim();
      }
      counter++;
      return `${counter}. ${output.toString()}`;
    })
    holder.push(upArr);
  });
}
```

```
})  
  
Logger.log(holder);  
range.setValues(holder);  
range.clearFormat();  
}
```

Custom Functions in Sheets

Google Sheets comes with 100s of built in functions

<https://support.google.com/docs/table/25273>

You can also create your own custom functions using Google Apps Script. Custom functions are created using standard JavaScript syntax within Google Apps Script. Custom functions never ask users to authorize access to personal data.

To use custom functions click on the cell, and just like any other custom function you can select it in the formula bar for within sheets. Start by typing = in the formula bar, followed by the function name and provide the required arguments for that function.

This can be values from other cells in the sheet. The Loading... will display in the sheet and once complete it will return the value in the cell. If there is an error in the input format that isn't expected in the function you will see the returned error in the cell.

When naming your custom functions in sheets, avoid the built-in function names, as well do not end the name with _ as this is reserved for private functions. The name of the function must be set as the function name in the Apps Script. Best practice is to capitalize the name although capitalization is not required, it does make it easier to read the functions in the code.

Custom functions can be selected and run from the formula bar, by selecting the cell you want to apply the function to and starting to type with the = equal sign then typing in the function name.

E2		<i>fx</i>	=MYCUSTOM(B1)		
	A	B	C	D	E
1	2	32	62	92	
2	3	23	43	63	1034
3	4	3	464	694	539

How to Create custom Number Functions

1. Create a function name which expects a numeric input value
2. Create the calculation of the data from the input, using the return to send the calculated results back into the cell.

```
function MYCUSTOM(val) {
  const rep = val * val + 10;
  return rep;
}
```

```
function MYDOUBLER(val) {
  return val * 2;
}
```

```
function MYTEST(val) {
  return val;
}
```

How to Create a custom function that uses the string values and calculates the length of the string.

1. Create a function that requires two parameters, both string values
2. Add the strings together , return the result
3. Add to the returned result the string length value

```
function FULLNAME(first,last){
  const val = `${first} ${last}`
  return `${val} ${val.length}`;
}
```

How to Create a custom function to calculate sales tax

1. Create the function with a numeric parameter
2. Multiply the input value by the value for the tax and return the results.

```
function SALESTAX(val) {
  return val * 0.15;
}
```

How to Create a custom function to return Latitude and Longitude of a string value location name.

1. Create a custom function that takes in one string argument
2. Using the Maps object creates a new geocoder. Maps.newGeocoder();
3. Use the geocoder and send the string value from the function into it. If there are results, select the latitude and longitude from the returned object for location data.
4. Return the custom string with Lat and Long back to the function return.

```
function LATLONGFIND(val){
  const geoCoder = Maps.newGeocoder();
  try {
    const loc = geoCoder.geocode(val);
    if(loc.status){
      const locat = loc.results[0].geometry.location;
```

```

    return `lat:${locat.lat} long:${locat.lng}`;
  }else{
    return 'Not Found';}
} catch(error) {
  return 'Not Found';
}
}

function test(){
  const geoCoder = Maps.newGeocoder();
  const val = 'Toronto';
  const loc = geoCoder.geocode(val);
  if(loc.status){
    Logger.log(loc.results[0].geometry.location);
    const locat = loc.results[0].geometry.location;
    Logger.log(`lat:${locat.lat} long:${locat.lng}`);
    return `lat:${locat.lat} long:${locat.lng}`;
  }else{
    return 'Not Found';
  }
}

```

Web Apps with Apps Script

Web apps allow you to publish your script to the web, with a unique URL that others can then access and interact with. Best practice is to use a standalone script for web apps, although you can also create the same webapp within a bound script. For the web app to return results it uses the default functions within Apps Script of either a doGet(e) or

doPost(e) function or both can be used on the same endpoint. The results will be returned depending on the method used to connect to the webapp. If you open the URL in a browser the method used is GET.

Using the HtmlService.createHtmlOutput() this will output the string content as html code into the web app.

The e argument represents an event parameter that contains information from the request URL parameters. For the Get method the parameters will be as below in the object response.

```
{"contextPath":"","parameter":{"id":"3"},"contentLength":-1,"parameters":{"id":["3"]},"queryString":"id=3"}
```

The value of parameter and parameters are both objects that will contain any value parameter. The parameters can be added to the URL using the ? and a key with a value separated by an equal sign. You can add additional parameters using the & symbol to the request URL.

```
https://script.google.com/macros/s/AKfycbxgspZ0QmgZDnPZVZo0glgg4CqjeF4cTwpcr1cLkZATu6WFsqQE887e6b1BGuLW99SCA/exec?id=3
```

Create a custom response with data coming from a Spreadsheet. Using the parameter of id, retrieve the corresponding row value for the id value.

1. Using doGet(e) retrieve the e parameters and run the code in the default function for the webapp
2. Using a condition check if the property value of id is contained in the e parameters object. If it is then update the output html with the content from the spreadsheet row.
3. Select the sheet with the data that you want to use.
SpreadsheetApp.openById(id).getSheetByName()
4. Using the getValues() retrieve the data in a nested array format. The row value can now be used to retrieve an item from the array, as each array nested within it

represents a row of data from the spreadsheet. Return the data of the resulting row.

5. Update the output html with the sheet data
6. Create the return of the HTMLService and create the HTML output.

```
function doGet(e) {
  let html = `

# Hello World 3</h1>`; const eData = JSON.stringify(e); if('id' in e.parameter){ let val = e.parameter['id']; let data = buildHTML(val); html += ` Found Row #${e.parameter['id']}</div>`; html += `${data[2]}</h2>`; } html += ` ${eData}</div>`; return HtmlService.createHtmlOutput(html); }


```

```
function test1(){
  const val = buildHTML(1);
  Logger.log(val[2]);
}
```

```
function buildHTML(row){
  row = row-1 || 0;
  const id = '1aSBcG-tw-zZzke1CGXIm465ky4ZcId9mg0MyXNk0Pzg';
```

```

const sheet =
SpreadsheetApp.openById(id).getSheetByName('Sheet2');
const data = sheet.getDataRange().getValues();
return data[row];
}

```

Do more with Web Apps

With web apps you can select the output content type.

Content Service - is ideal for outputting straight text content or MIME type content like JSON data. Content Service does not wrap the container with the iframe object like the HTML service does. Depending on how you want to access the data and the type of data Content Service vs the HTMLService will both output web app results.

```

function doGet() {
  const output = 'Hello World';
  return ContentService.createTextOutput(output);
}

```

Using the content service output JSON data into the web app.

1. Create an object in your app script
2. Using JSON.stringify(myObj) convert the object data into a string value
3. Set the Mimetype for the returned results.
 .setMimeType(ContentService.MimeType.JSON);
4. Return the object within the ContentService as text output.
 ContentService.createTextOutput()

```

function doGet() {
  const myObj = [{

```

```

    first : "Laurence",
    last  : "Svekis",
    id    : 1000
  }, {
    first : "John",
    last  : "Test",
    id    : 20
  }
];
return
ContentService.createTextOutput(JSON.stringify(myObj)).setMimeType(
ContentService.MimeType.JSON);
}

```

With Apps Script you can create HTML as a string value and send it to the webapp as output.

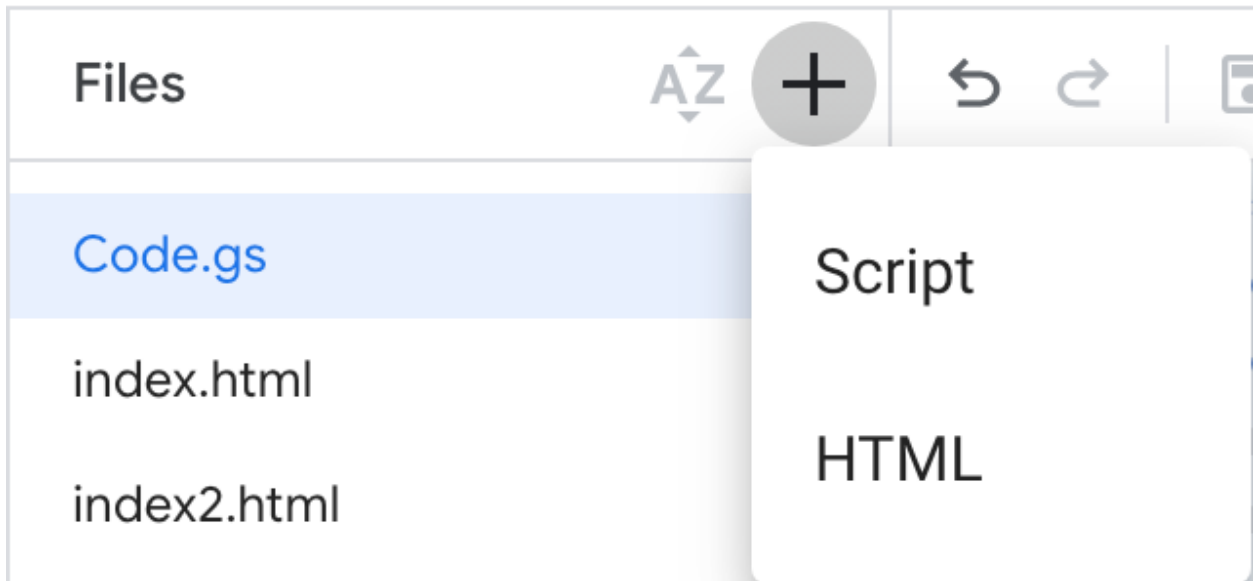
```

function doGet() {
  const html = `

# 


```

Within the HtmlService you can also create html code from a file, the file is in the Apps Script menu under files. To create a new file, press the + in the files menu and select HTML file. Give the file a name, you don't need to include the .html extension.



Using the HtmlService you can select the file and generate the output of the file as HTML.

```
function doGet() {  
  return HtmlService.createHtmlOutputFromFile('index');  
}
```

HTML file contents (index.html)

```
<!DOCTYPE html>  
<html>  
  <head>  
    <base target="_top">  
  </head>  
  <body>  
    <h1>Laurence Svekis</h1>  
    Hello World 2
```

```
</body>
</html>
```

How to use templated HTML

Render HTML results from server side Apps Script code. Using the Scriptlets. To use scriptlets, use the syntax `<?= ... ?>` this will output the results of their code into the client side html page.

Using the scriptlet tags output the value of the variable from Google Apps Script.

1. Create a variable with the name val.
2. Create the output object from the template html code. `const output = HtmlService.createTemplate(html);`
3. Create some html code, you can also create code within the script tags as this will be evaluated and rendered into the web app page. Using the scriptlet syntax output the value of the variable into the client side code. Assign a value to the variable val into the output object.
4. Using the variable from the Apps Script you can hard code it directly within the HTML output that is being created, as this is expecting a string value.
5. Return the template output using `evaluate()` to construct the output.

```
function doGet() {
  const val = 'My New Value Laurence 1';
  let html = `

# Laurence Svekis 4</h1>`; html += `<?= val ?></h2>`; html += `


```

```

output.val = 'Laurence Svekis Value 2';
return output.evaluate();
}

```

How to use Apps Script code and functions directly into the client side code.

Within the web app output, you can use the scriptlets to get output object values from the server side. Assign the values to the output object in order to be able to retrieve them within the client side code. You can also invoke functions from the client side to return data within the Google Apps Script server side code.

1. Create an html file called index2. Within the file use the scriptlet to assign a value to data variable from the response data coming from a function on google script side called fromSheet().
2. Within the Apps Script code, create a function called fromSheet() which connects to a spreadsheet, gets the values from the sheet data and returns the array of data.
3. Back in the client side index2 file, using for each loop through the returned data from the fromSheet() function.
4. Output a value from the output object directly in the template using `<?= ?>`
5. Using the scriptlet add several numbers together to produce an output value.
`<?= 5 + 5 + 510 ?>`
6. Using a condition with a value that comes from the output object, apply the different output results depending on the condition.
7. Send an object into the output data, loop through the array of data outputting the results into the client side page within the scriptlet.

```

function doGet() {
  const boo = true;
  const output = HtmlService.createTemplateFromFile('index2');
  output.val = 'Laurence Svekis Value 2';
  output.boo = boo;
}

```

```

output.myObj = [{
  first : "Laurence",
  last  : "Svekis",
  id    : 1000
},{
  first : "John",
  last  : "Test",
  id    : 20
}]
return output.evaluate();
}

```

```

function fromSheet(){
  const id = '1aSBcG-tw-zZzke1CGXIm465ky4ZcId9mg0MyXNk0Pzg' ;
  const data =
  SpreadsheetApp.openById(id).getSheetByName('Sheet2').getDataRange().getValues();
  return data;
}

```

HTML file contents (index2.html)

```

<!DOCTYPE html>
<html>
  <head>
    <base target="_top">
  </head>
  <body>

```



```

<? const data = fromSheet(); ?>
<ol>
<? data.forEach((el)=>{ ?>
    <li><?= el.join(' ') ?></li>
<? }) ?>
<ol>
<h2><?= val ?></h2>
<?= 5 + 5 + 510 ?>
<? if (boo) { ?>
    <h1>The result was true</h1>
<? } else {?>
    <h1>The result was false</h1>
<? } ?>
<ul>
<? myObj.forEach((row)=>{ ?>
    <li><?= `${row.first} ${row.last}` ?></li>
<? }) ?>
</ul>
</body>
</html>

```

doPost and doGet as an endpoint for web applications.

You can setup your webapp to serve as an endpoint for web applications. In this example we explore how to make both GET method requests and POST method

requests to the endpoint, retrieving and returning data from a spreadsheet into the JavaScript code as data from an AJAX request.

In this lesson I use Visual Studio Code as my editor. <https://code.visualstudio.com/>

Also I use LiveServer addon for Visual Studio Code to output the webpage as http locally.

1. Create an HTML file that has an input field, a button and an element that can be used to output HTML into from the JavaScript code.
2. Create the doGet() function in your Apps Script code. Connect to a spreadsheet with some data that you want to use. Get all the values from the spreadsheet and build an object that can be returned from the sheet data and output into the web app as JSON data.
3. Use the ContentService to output the webapp content as JSON data, which then can be used by the JavaScript application to connect to the endpoint. return ContentService.createTextOutput(JSON.stringify(holder)).setMimeType(ContentService.MimeType.JSON);
4. Deploy the web app, and get the exec URL from the deployment to use in your JavaScript Code.
5. Create a JavaScript file that uses the webapp Exec URL with fetch. Select the DOM page elements as object, attach an eventlistener to the button on the page.
6. When the button is clicked invoke the function that will make the fetch request to the web app endpoint URL, retrieve the JSON data and output it to the page using JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Course</title>
  </head>
  <body>
    <input type="number" value='1'>
```

```

    <button>Check</button>
    <div class="output">Output</div>
    <script src="gas.js"></script>
  </body>
</html>

```

```

function doGet(e) {
  const id = '1aSBcG-tw-zZzke1CGXIm465ky4ZcId9mg0MyXNk0Pzg';
  const sheet =
SpreadsheetApp.openById(id).getSheetByName('Sheet2');
  const data = sheet.getDataRange().getValues();
  const holder = [];
  data.forEach((row)=>{
    const temp = {
      first : row[0],
      last : row[1],
      fullName : row[2]
    }
    holder.push(temp);
  })
  Logger.log(holder);
  return
  ContentService.createTextOutput(JSON.stringify(holder)).setMimeType(
ContentService.MimeType.JSON);
}

```

```

const url =
'https://script.google.com/macros/s/AKfycbxBHUXc5rPxG2TPYNSFFGBT
Eaoz5WDj0e7WiHhjeKQ_P03RtI94X5t3IN8IBvrrEFgktw/exec';
const output = document.querySelector('.output');
const btn = document.querySelector('button');
btn.addEventListener('click', (e) => {
    fetch(url)
        .then(res => res.json())
        .then(data => {
            addData(data);
        })
});

function addData(data) {
    output.innerHTML = '';
    data.forEach((row) => {
        output.innerHTML += `<div>${row.first} ${row.last}
${row.fullName}</div>`;
    })
}

```

How to use a Web App and make a POST request with JavaScript fetch method to get JSON data back to a web application.

In this example we will connect to a web app URL endpoint with AJAX, sending a value from the input field for the row of data that will be retrieved from the endpoint. This row value can be changed within the input field selecting values from different selected rows coming from a spreadsheet.

1. Create the POST method in Apps Script to output the values of the spreadsheet data. Select from the e parameters the value of the row to return.
2. If there is a value for row, use this value to get the response back for the spreadsheet matching row data. To debug you can return the entire e object as a stringified value so that you can see the response in the web application. This content can be difficult to debug since the exec needs to be redeployed on all changes and the dev cannot be used to see the response values.
3. Update the JavaScript code to get the input value from the input field. Add a UL list to the output element that can then be used to add list items into. With JavaScript you can add page elements using the document.createElement() method. To append them to other existing element use the append() or appendChild() methods.
4. Create a new FormData object in JavaScript `const formData = new FormData();`
5. Add to the formData object the values of the input field under a property name row. `formData.append('row',myInput.value);`
6. Send the fetch request using the POST method and attach the body contents from the formData object. `fetch(url,{ method:'POST', body: formData })`
7. Add the response value from the AJAX request to the webpage as a new list item `document.createElement('li')`

```
function doPost(e) {
  const id = '1aSBcG-tw-zZzke1CGXIm465ky4ZcId9mg0MyXNk0Pzg';
  const sheet =
SpreadsheetApp.openById(id).getSheetByName('Sheet2');
  const data = sheet.getDataRange().getValues();
  const holder = [JSON.stringify(e)];
  if('row' in e.parameter){
    let val = e.parameter['row'];
    val--;
    let row = data[val];
```

```

const temp = {
  first : row[0],
  last : row[1],
  fullName : row[2]
}
holder.push(temp);
}
return
ContentService.createTextOutput(JSON.stringify(holder)).setMimeType(ContentService.MimeType.JSON);
}

```

Update the JavaScript code with the below :

```

const myInput = document.querySelector('input');
const ul = document.createElement('ul');
output.innerHTML = '';
output.append(ul);
btn.addEventListener('click', (e) => {
  const formData = new FormData();
  formData.append('row', myInput.value);
  fetch(url, {
    method: 'POST',
    body: formData
  })
    .then(res => res.json())
    .then(data => {
      console.log(data[1]);
      addOutput(data[1]);
    })
});

```

```

    })
  })
  function addOutput(row){
    const li = document.createElement('li');
    ul.append(li);
    li.innerHTML = `<div>${row.first} ${row.last}
    ${row.fullName}</div>`;
  }

```

ClientSide to ServerSide WebApp

Google web apps can run client side code that can easily be used to connect to server side Apps Script functionality. To execute server-side functions from client-side code, use `google.script.run`. `google.script.run` is an asynchronous client-side JavaScript API available in web app HTML-service pages that can call server-side Apps Script functions.

In this example we will demonstrate how to send data objects from the Apps Script server side into the client side and use the data object within JavaScript. Also how we can send data from the client side input field values to the server side script to then be used to update and append content into a selected spreadsheet.

1. Setup the `doGet()` method to create the web app output page. Create an object with some data and add it to the html object that is created from the template file `index` `HtmlService.createTemplateFromFile('index')`
2. Create the template file `index` in the file menu of the Google Apps Script editor.
3. Add the data object from the server side using a scriptlet adding it into a Javascript variable called `data`. `const data = <?!= JSON.stringify(data) ?>;`

4. Create page elements, and select those with JavaScript. Create 2 input fields, and a button that can be used to invoke the function call sending data to the server side function. Add an event listener to the button
5. When the button is pressed, gather the input field data and create an object that will be the argument of the google script function sending to the function on the server side.
6. Using the google script service run a function in the Apps Script called testFun sending the data from the frontend to it.
`google.script.run.withSuccessHandler(onSuccess).testFun(temp);`
7. Create a function that will receive the response data on the client side, the function that was used in the withSuccessHandler argument. Add an update to the page.
8. Get the input field values and create a function called testFun() on the server side within the google Apps Script. Connect to a spreadsheet and append the row of data to the sheet. Get the value of the last row and return that to the client side within the return response value.

```

<script>
  const data = <?!= JSON.stringify(data) ?>;
  console.log(data);
</script>
<!DOCTYPE html>
<html>
  <head>
    <base target="_top">
  </head>
  <body>
    <h1></h1>

```



```

<input type="text" name="first" value="Laurence">
<input type="text" name="last" value="Svekis">
<button>Add to Sheet</button>
<div class="output"></div>
<script>
  const h1 = document.querySelector('h1');
  const btn = document.querySelector('button');
  const myFirst =
document.querySelector('input[name="first"]');
  const myLast =
document.querySelector('input[name="last"]');
  h1.textContent = data.data.first;
  btn.addEventListener('click', (e)=>{
    const temp = {
      first : myFirst.value,
      last : myLast.value
    };

google.script.run.withSuccessHandler(onSuccess).testFun(temp);
  })

  function onSuccess(res){
    console.log(res);
    const html = `Item added to row #${res}`;
    document.querySelector('.output').textContent = html;
  }
</script>

```

```

</body>
</html>

```

Client side index.html

```

<script>
  const data = <?!= JSON.stringify(data) ?>;
  console.log(data);
</script>
<!DOCTYPE html>
<html>
  <head>
    <base target="_top">
  </head>
  <body>
    <h1></h1>
    <input type="text" name="first" value="Laurence">
    <input type="text" name="last" value="Svekis">
    <button>Add to Sheet</button>
    <div class="output"></div>
    <script>
      const h1 = document.querySelector('h1');
      const btn = document.querySelector('button');
      const myFirst =
document.querySelector('input[name="first"]');
      const myLast =
document.querySelector('input[name="last"]');
      h1.textContent = data.data.first;

```

```

btn.addEventListener('click', (e)=>{
  const temp = {
    first : myFirst.value,
    last : myLast.value
  };

  google.script.run.withSuccessHandler(onSuccess).testFun(temp);
})

function onSuccess(res){
  console.log(res);
  const html = `Item added to row #${res}`;
  document.querySelector('.output').textContent = html;
}
</script>
</body>
</html>

```

Google Apps Script Triggers for Automation

Triggers allow you to run a function automatically within Apps Script. There are simple triggers that are built in and use custom functions in order to start and invoke the function. Within the event object of each event, the triggered function contains an event object that has information about the context in which the event occurred. Some commonly used simple triggers are `onOpen()` which runs when the spreadsheet, document, is opened. There is `onEdit()` that runs when the user changes a value in a spreadsheet. Also for web apps `doGet()` and `doPost()` are simple triggers that get invoked when the user visits the web application on either method.

Simple triggers must be in a bound script in order to run the application triggers, they also fire automatically without asking for user permissions. They will not run if the file is opened in read only mode. They are also limited in some services that would require authorization. They cannot run for longer than 30 seconds. Simple triggers are subject to Apps Script trigger quota limits.

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

```
function test1(){  
  output('test1');  
}
```

```
function test2(){  
  output('test2');  
}
```

```
function output(val){  
  SpreadsheetApp.getUi().alert(val);  
}
```

```
function onEdit(e){  
  output(JSON.stringify(e));  
}
```

```
}
```

Installable triggers provide more flexibility for users, they let Apps Script run a function automatically and can also run services that require authorization. Just like the simple triggers the event object is included in each event, that contains information about the context of the event. Installable triggers always run under the account of the person who created them. Installable triggers are subject to Apps Script trigger quota limits.


You can create installable triggers with Apps Script code.

```
function createTimers(){
  ScriptApp.newTrigger('logMe')
    .timeBased()
    .everyHours(4)
    .create();
}
```

Create a function that you can add to run automatically with a time based trigger.

1. Create a function that opens a spreadsheet, and generates a random value that can get added to the sheet.
2. Create the trigger either manually or with code to run every 1 minute.
3. Close the application and reopen after several minutes. Confirm the function ran and updated your Spreadsheet.
4. Open the Triggers menu option, and delete the trigger.

```
function logMe(){
  const id = '13UFJVM0ACo9mTDjNJnt_HBAJRcNtTYPwW-Wc0d0qU_s';
  const ss = SpreadsheetApp.openById(id).getSheetByName('log');
  const ran = Math.floor(Math.random()*1000);
  ss.appendRow(['test', ran]);
}
```


Apps Script
Test Au

ⓘ	Files	AZ +
<>	Code.gs	
🕒	Libraries	+
☰▶	Services	+
⚙️		