Apps Script Coding Examples

Google Apps Script is a cloud-based platform that allows users to extend Google’s G Suite of productivity tools with custom code. It is a JavaScript-based scripting language that runs on Google's servers and provides a seamless way to automate tasks, integrate various Google services, and build custom applications.

Google Apps Script integrates with several Google services such as Google Sheets, Google Forms, Gmail, Google Drive, and more, making it a versatile platform for automating tasks, streamlining workflows, and developing custom applications. For example, you can use Google Apps Script to:

Automatically generate reports from Google Sheets data
Automate email responses in Gmail
Pre-fill Google Forms with data from a spreadsheet
Organize files in Google Drive based on specific criteria
Google Apps Script has a user-friendly interface and a simple syntax, making it easy for both experienced and novice developers to get started. It also has several built-in functions and libraries that allow you to perform various tasks, such as sending emails, accessing data in spreadsheets, and more.

Another key feature of Google Apps Script is its ability to run in the cloud, so your scripts are always available and accessible from any device. You can also set up triggers to automatically run your scripts at specified intervals, such as daily, weekly, or monthly.

In summary, Google Apps Script is a cloud-based platform that provides a simple and powerful way to automate tasks, integrate Google services, and build custom applications. With its user-friendly interface, powerful scripting language, and seamless integration with G Suite, it's a great tool for anyone looking to increase their productivity and automate repetitive tasks.

Top Tips for Google Apps Script

10 tips to help you improve your Google Apps Script coding:

1. Use the Google Apps Script Debugger: Google Apps Script has a built-in debugger that can help you find and fix errors in your code.
2. Test your code with sample data: Before you run your code on real data, test it with a small set of sample data to make sure it works as expected.
3. Use proper naming conventions: Naming your variables and functions in a consistent and descriptive manner will make your code easier to read and understand.
4. Comment your code: Adding comments to your code will help you (and others) understand what your code is doing, especially if it's complex.
5. Use arrays and objects effectively: Arrays and objects are powerful data structures in Google Apps Script. Learn how to use them effectively to simplify your code and make it more readable.
6. Use built-in functions and methods: Google Apps Script has many built-in functions and methods that you can use to perform common tasks. Take advantage of these to save time and make your code more efficient.
7. Avoid hardcoding values: Avoid hardcoding values in your code, such as cell references or URLs, as this can make your

code difficult to maintain. Instead, use variables and functions to make your code more flexible.

8. Use error handling: Use error handling techniques to handle unexpected errors and keep your code running smoothly.
9. Write modular code: Write modular code by breaking your code into small, reusable functions that can be used in multiple places.
10. Keep your code organized: Keeping your code organized by using proper indentation, separating code into different functions and sections, and using meaningful variable names will make it easier to read, understand, and maintain.

Converting a Google Document to plain text

Converting a Google Document to plain text:

```
function convertToPlainText() {
  // Get the active document
  const document = DocumentApp.getActiveDocument();

  // Get the body of the document
  const body = document.getBody();

  // Get the plain text representation of the document
  const plainText = body.getText();
  // Log the plain text to the console
  Logger.log(plainText);
}
```

In this example, the convertToPlainText() function uses the DocumentApp class to convert a Google Document to plain text. The DocumentApp.getActiveDocument() method is used to get the active document, and the document.getBody() method is used to get the body of the document. The body.getText() method is then used to get the plain text representation of the document, which is logged to the console using the Logger.log() method.

Send the text version of the Doc using Apps Script to the email address of the apps script account

```
function converterText(){
  const doc = DocumentApp.openById(ID);
  const body = doc.getBody();  //DocumentBodySection
  const txt = body.getText();
  const email = Session.getActiveUser().getEmail();
  //Logger.log(email);
  MailApp.sendEmail(email,'My Doc Text',txt);
}
```

Google Sheets add numbers from columns

Google Apps Script that adds two numbers and displays the result in a Google Sheets cell:

```
function sumTwoNumbers() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var cell1 = sheet.getRange("A1").getValue();
  var cell2 = sheet.getRange("B1").getValue();
  var result = cell1 + cell2;
  sheet.getRange("C1").setValue(result);
}
```

Explanation:

function sumTwoNumbers(): This is the declaration of the function sumTwoNumbers, which contains the code that will be executed.

var sheet = SpreadsheetApp.getActiveSheet();: This line creates a variable sheet and assigns the active sheet to it using the SpreadsheetApp.getActiveSheet() method.

var cell1 = sheet.getRange("A1").getValue();: This line creates a variable cell1 and assigns the value of cell A1 to it using the getValue() method.

var cell2 = sheet.getRange("B1").getValue();: This line creates a variable cell2 and assigns the value of cell B1 to it using the getValue() method.

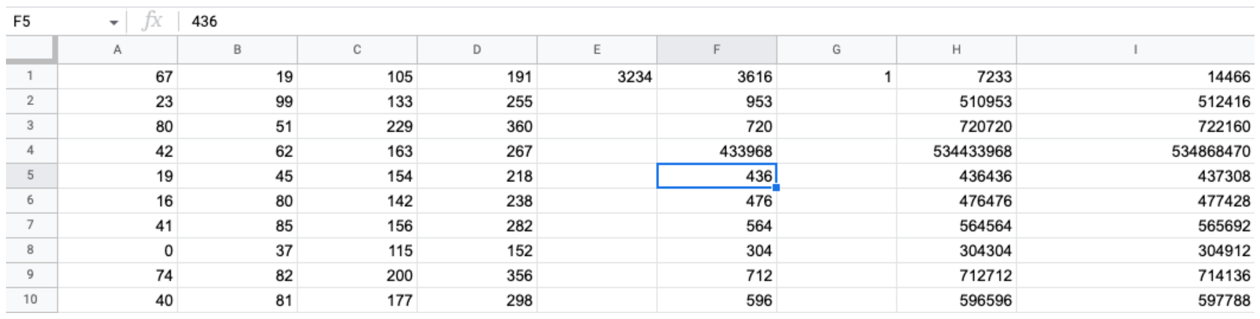
var result = cell1 + cell2;: This line creates a variable result and assigns the sum of cell1 and cell2 to it.

sheet.getRange("C1").setValue(result);: This line sets the value of cell C1 to the value of result using the setValue() method.

To run this script, you would need to open a Google Sheet and then open the Apps Script editor by clicking on the "Tools" menu and selecting "Script editor." Then, paste the code into the editor, save the script, and run the sumTwoNumbers() function.

Sheet Data and Cell Values

Adding numbers together, creating random numbers to populate the sheet data.



F5 436

	A	B	C	D	E	F	G	H	I
1	67	19	105	191	3234	3616	1	7233	14466
2	23	99	133	255		953		510953	512416
3	80	51	229	360		720		720720	722160
4	42	62	163	267		433968		534433968	534868470
5	19	45	154	218		436		436436	437308
6	16	80	142	238		476		476476	477428
7	41	85	156	282		564		564564	565692
8	0	37	115	152		304		304304	304912
9	74	82	200	356		712		712712	714136
10	40	81	177	298		596		596596	597788

```
const ID = '132trk8qIk';

function adder(){
  const sheet =
SpreadsheetApp.openById(ID).getSheetByName('add');
  const data = sheet.getDataRange().getValues();
  data.forEach((row,index) =>{
    let total = 0;
    row.forEach(cell => {
      if(cell){
      total = total + parseInt(cell);
      }
    })
    const rowValue = index+1;
    const range =
sheet.getRange(rowValue,row.length+1,1,1);
    range.setValue(total);
  })
}
```

```
function makeNums(){
  const ss = SpreadsheetApp.openById(ID);
  const sheet = ss.getSheetByName('add');
  for(let i=0;i<20;i++){
    const arr = [getRan(),getRan(),getRan()];
    sheet.appendRow(arr);
  }
  Logger.log(sheet);
}

function getRan(){
  return Math.floor(Math.random()*100);
}
```

Add Image to Sheet Cell

add an image to a cell in a Google Sheets spreadsheet using Google Apps Script:

```
function addImageToCell() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var imageUrl = "https://www.example.com/image.png";
  var img = UrlFetchApp.fetch(imageUrl).getBlob();

  sheet.insertImage(img, 1,1);
}
```

Explanation:

First, we get a reference to the active sheet using SpreadsheetApp.getActiveSheet().

Next, we get a reference to the cell where we want to insert the image using sheet.getRange("A1"). You can specify any cell you want by changing the argument to getRange().

Then, we specify the URL of the image we want to insert using the imageUrl variable.

We use the UrlFetchApp.fetch() method to retrieve the image data from the specified URL. This method returns a Response object, which we get the binary data (the image) from using getBlob().

Finally, we use the sheet.insertImage() method to insert the image into the specified cell. The first argument is the image data (the blob we obtained in step 4), and the second argument is the cell reference.

```
function imageAdder(){
  const ss = SpreadsheetApp.openById(ID);
  const sheet = ss.getSheetByName('my');
  const imageURL =
'http://www.discoveryvip.com/img/d.png';
  const img = UrlFetchApp.fetch(imageURL).getBlob();
  sheet.insertImage(img,20,20);
  Logger.log(img);
}
```

New CellImage in Sheets

To add an image into a specific cell this can be done using the newCellImage method.

```
function imageAdderCell(){
  const ss = SpreadsheetApp.openById(ID);
  const sheet = ss.getSheetByName('my');
  const range = sheet.getRange(1,1,3,3);
  const imageURL =
'http://www.discoveryvip.com/img/d.png';
```

```
  const image = SpreadsheetApp
  .newCellImage()
  .setSourceUrl(imageURL)
  .setAltTextTitle('discoveryvip')
  .build();
  range.setValue(image);
}
```

Logging to the console

basic syntax and structure of a Google Apps Script.

```
function helloWorld() {
  // This line logs "Hello, world!" to the console
  Logger.log("Hello, world!");
}
```

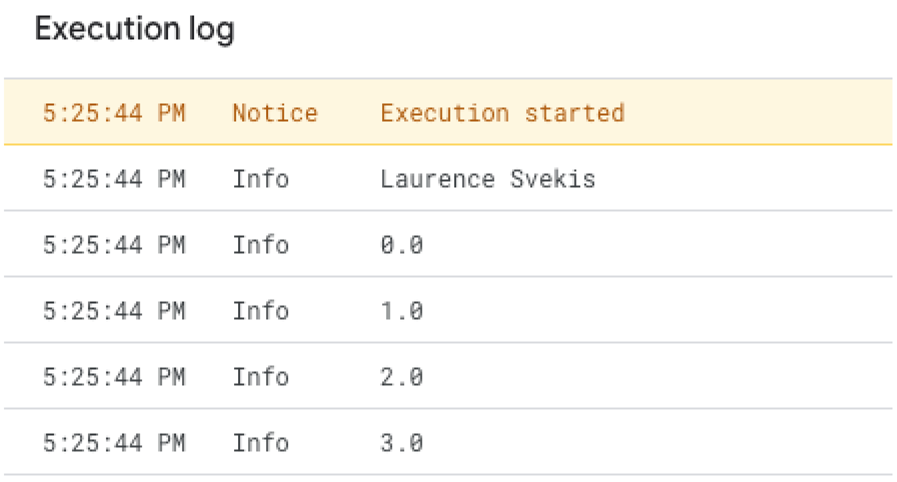
This is a simple function named helloWorld that logs the string "Hello, world!" to the Google Apps Script logging console.

Here's what's happening in this example:

The function keyword declares that this is a function.
The helloWorld function name follows the function keyword.
The code inside the curly braces {} is the body of the function.
The Logger.log statement logs the message "Hello, world!" to the console.
You can run this script by selecting the code and clicking the "Run" button in the Google Apps Script editor, or by pressing Ctrl + Enter (Cmd + Enter on a Mac). When you run this script, the message "Hello, world!" will appear in the logging console.

This is just a simple example, but Google Apps Script provides a wide range of functionality that you can use to automate tasks, manipulate data, and interact with other Google Workspace services. With a little bit of code, you can streamline your work and save yourself a lot of time.

```
function hello(){
  const message = 'Laurence Svekis';
  Logger.log(message);
  for(let i=0;i<20;i++){
    Logger.log(i);
  }
}
```



Execution log

```
5:25:44 PM   Notice   Execution started
5:25:44 PM   Info     Laurence Svekis
5:25:44 PM   Info     0.0
5:25:44 PM   Info     1.0
5:25:44 PM   Info     2.0
5:25:44 PM   Info     3.0
```

Adding numbers and showing an alert

Sheets Add Numbers UI Alert.

Google Apps Script that adds up two numbers and displays the result in a dialog box:

```
function addNumbers() {
  var num1 = 5;
  var num2 = 3;
  var result = num1 + num2;
  SpreadsheetApp.getUi().alert("The result is: " +
result);
}
```

Explanation:

The function addNumbers is defined.

Two variables num1 and num2 are defined and assigned values of 5 and 3 respectively.
The result of num1 + num2 is calculated and stored in the variable result.
The SpreadsheetApp.getUi().alert method is used to display a dialog box with the text "The result is: 8", which is the value of the result variable.
This script can be run in a Google Spreadsheet by opening the script editor (under the "Tools" menu), copying and pasting the code into the editor, and then clicking the "Run" button

```
function showAl(){
  const num1 = 100;
  const num2 = 144;
  const result = num1 + num2;
  const output = `The result of ${num1} + ${num2} =
${result}`;
  SpreadsheetApp.getUi().alert(output);
}
```

Read Data from from Sheet

Google Apps Script that reads data from a Google Sheet, performs calculations on the data, and writes the results back to the sheet:

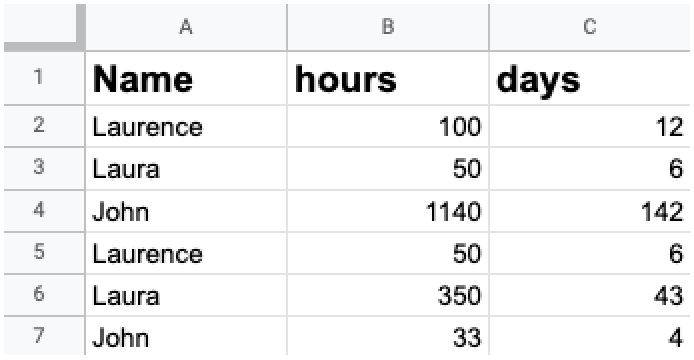
```
function calculateAverages() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();

  var total = 0;
  for (var i = 0; i < data.length; i++) {
    total += data[i][0];
  }
  var average = total / data.length;

  sheet.getRange(data.length + 1, 1).setValue(average);
}
```

Explanation:

The function calculateAverages is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.

A for loop is used to sum up the values in the first column of the data array. The total variable is used to keep track of the sum.
The average value is calculated by dividing the total by the number of rows in the data (data.length).
The average value is written to the next empty row in the first column of the sheet using the sheet.getRange(data.length + 1, 1).setValue method.
This script can be run in a Google Spreadsheet by opening the script editor (under the "Tools" menu), copying and pasting the code into the editor, and then clicking the "Run" button. It will read the data in the sheet, perform the calculation, and write the result to the next empty row in column A.

Get sheet data - apply changes and set value in sheet data



	A	B	C
1	**Name**	**hours**	**days**
2	Laurence	100	12
3	Laura	50	6
4	John	1140	142
5	Laurence	50	6
6	Laura	350	43
7	John	33	4

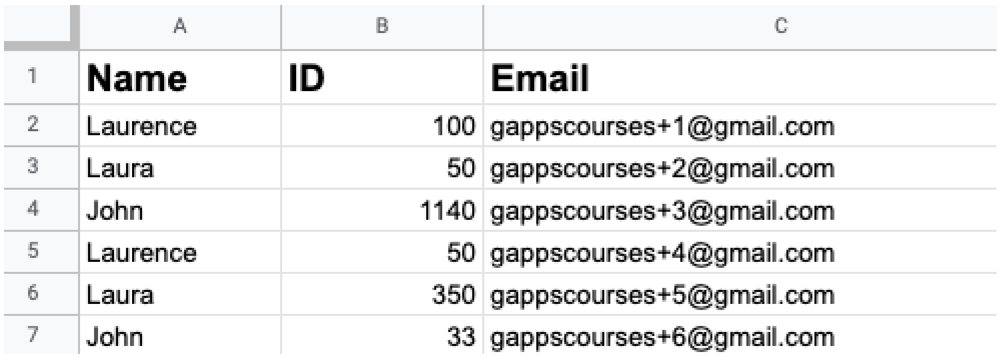
```
function sheetData2(){
  const sheet =
SpreadsheetApp.openById(ID).getSheetByName('data');
  const data = sheet.getDataRange().getValues();
  const headings = data[0];
  const results = data.slice(1);
  results.forEach((row,index) => {
    const days = (Math.floor(parseInt(row[1])/8));
    sheet.getRange(index+2,3).setValue(days);
  })
  //Logger.log(results);

}
```

Send email from Sheet

Google Apps Script that sends an email based on the contents of a Google Sheet:

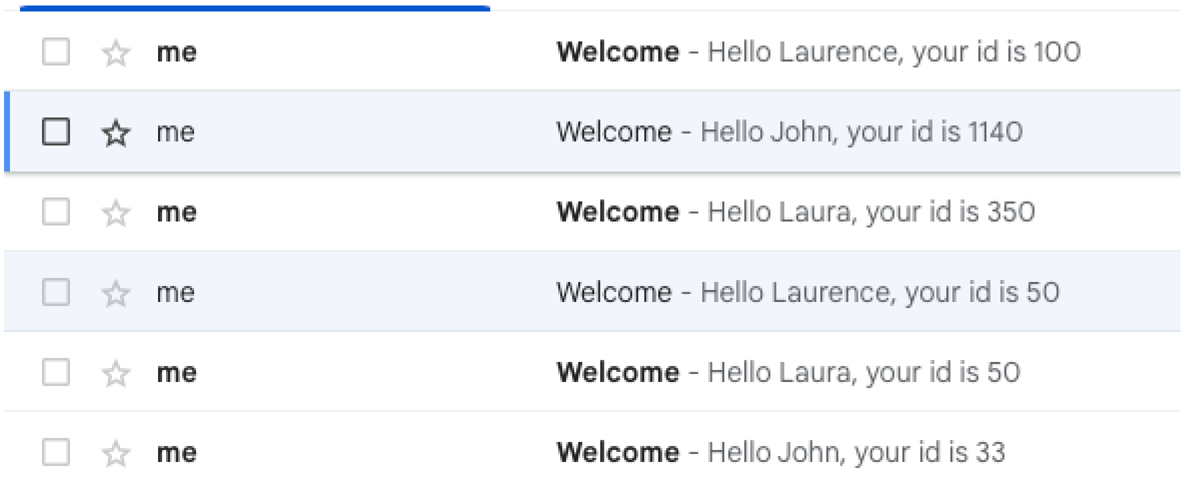
```
function sendEmails() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();

  for (var i = 1; i < data.length; i++) {
    var emailAddress = data[i][0];
    var message = data[i][1];
    var subject = "Important Message";

    MailApp.sendEmail(emailAddress, subject, message);
  }
}
```

Use Sheet data for custom emails out



	A	B	C
1	**Name**	**ID**	**Email**
2	Laurence	100	gappscourses+1@gmail.com
3	Laura	50	gappscourses+2@gmail.com
4	John	1140	gappscourses+3@gmail.com
5	Laurence	50	gappscourses+4@gmail.com
6	Laura	350	gappscourses+5@gmail.com
7	John	33	gappscourses+6@gmail.com

```
function senderEmail(){
  const sheet = SpreadsheetApp
              .openById(ID)
              .getSheetByName('Users');
  const data = sheet.getDataRange().getValues();
  const users = data.slice(1);
  Logger.log(users);
  users.forEach((user,index)=>{
    const userName = user[0];
    const email = user[2];
    const id = user[1];
    const message = `Hello ${userName}, your id is
${id}`;
    MailApp.sendEmail(email,'Welcome',message);
  })

}
```



me Welcome - Hello Laurence, your id is 100
me Welcome - Hello John, your id is 1140
me Welcome - Hello Laura, your id is 350
me Welcome - Hello Laurence, your id is 50
me Welcome - Hello Laura, your id is 50
me Welcome - Hello John, your id is 33

Explanation:

The function sendEmails is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.

The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.
A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).
For each row, the email address is stored in the emailAddress variable, the message text is stored in the message variable, and the subject of the email is set to "Important Message".
The mailApp.sendEmail method is used to send an email to the address stored in emailAddress, with the subject and message stored in subject and message respectively.
This script can be run in a Google Spreadsheet by opening the script editor (under the "Tools" menu), copying and pasting the code into the editor, and then clicking the "Run" button. It will read the data in the sheet, send an email to each email address specified in column A, with the message specified in column B.

Send Emails to recipients from Sheet data

Google Apps Script that sends an email to a list of recipients based on data in a Google Sheet:

```
function sendEmails() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();

  for (var i = 1; i < data.length; i++) {
    var email = data[i][0];
    var subject = data[i][1];
    var message = data[i][2];

    MailApp.sendEmail(email, subject, message);
  }
}
```

Explanation:

The function sendEmails is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.
A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).
For each row, the email address, subject, and message are stored in the email, subject, and message variables respectively.

An email is sent to the recipient specified by email using the MailApp.sendEmail method, with the subject and message specified by subject and message respectively.

Send emails from User list in Sheets

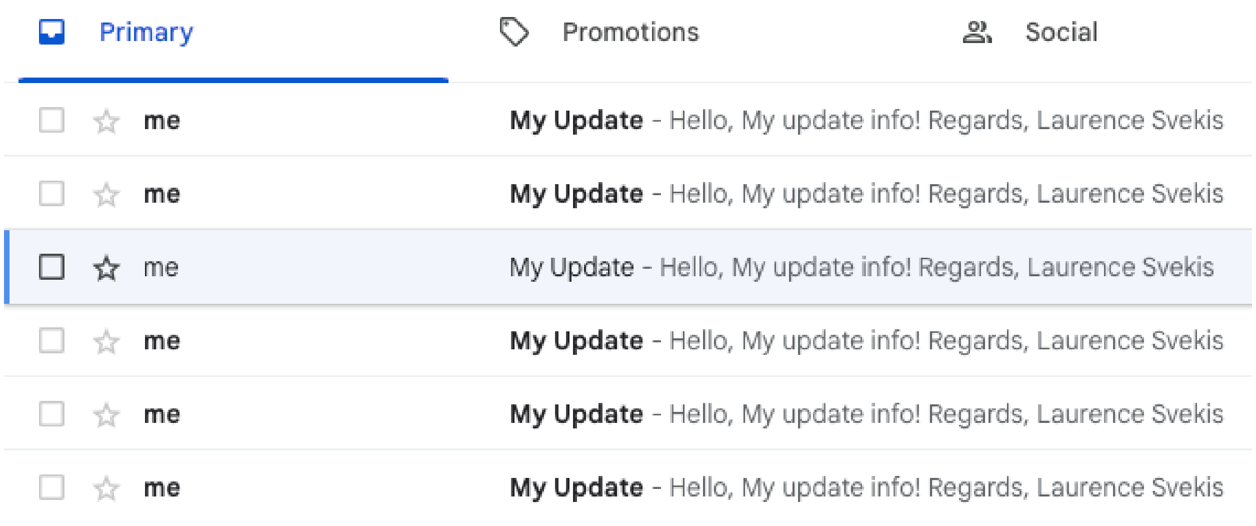
Google Apps Script that sends an email to a list of recipients from a Google Sheet:

```
function sendEmails() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();
  var subject = "Important Update";
  var message = "Hello,\n\nThis is an important update.
Please read carefully.\n\nBest regards,\nYour Name";

  for (var i = 1; i < data.length; i++) {
    var email = data[i][0];
    MailApp.sendEmail(email, subject, message);
  }
}
```

Explanation:

The function sendEmails is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.
The subject and message variables are set to the subject and message of the email respectively.
A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).
For each row, the email address is stored in the email variable.
An email is sent to the recipient using the MailApp.sendEmail method, with the recipient's email address, subject, and message specified by email, subject, and message respectively.

Sending an email to all users from the spreadsheet data



me My Update - Hello, My update info! Regards, Laurence Svekis
me My Update - Hello, My update info! Regards, Laurence Svekis
me My Update - Hello, My update info! Regards, Laurence Svekis
me My Update - Hello, My update info! Regards, Laurence Svekis
me My Update - Hello, My update info! Regards, Laurence Svekis
me My Update - Hello, My update info! Regards, Laurence Svekis

```
function emailOut(){
  const sheet = SpreadsheetApp.getActiveSheet();
  const data = sheet.getDataRange().getValues();
  Logger.log(data);
  const subject = 'My Update';
  const message = 'Hello, \n\nMy update
info!\n\nRegards,\nLaurence Svekis';
  for(let i=1;i<data.length;i++){
    const email = (data[i][2]);
    MailApp.sendEmail(email,subject,message);
  }
}
```

Apps Script on Sheets examples

Google Apps Script that performs various operations on a Google Sheet, including formatting cells, calculating values, and creating charts:

```
function formatSheet() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();
  // Format the header row in bold
  var headerRange = sheet.getRange(1, 1, 1,
data[0].length);
  headerRange.setFontWeight("bold");

  // Add borders to all cells
  var borderRange = sheet.getRange(1, 1, data.length,
data[0].length);
  borderRange.setBorder(true, true, true, true, null,
null);

  // Calculate the sum of the values in column 1
  var sum = 0;
  for (var i = 1; i < data.length; i++) {
    sum += data[i][0];
  }

  // Write the sum to the bottom of column 1
  sheet.getRange(data.length + 1, 1).setValue(sum);

  // Create a chart to visualize the data
  var chart = sheet.newChart()
      .setChartType(Charts.ChartType.BAR)
      .addRange(sheet.getRange(1, 1, data.length, 1))
      .setOption('title', 'Column 1 Data')
      .setOption('width', 400)
      .setOption('height', 300)
      .setPosition(5,6,0,0)
      .build();
  sheet.insertChart(chart);
}
```

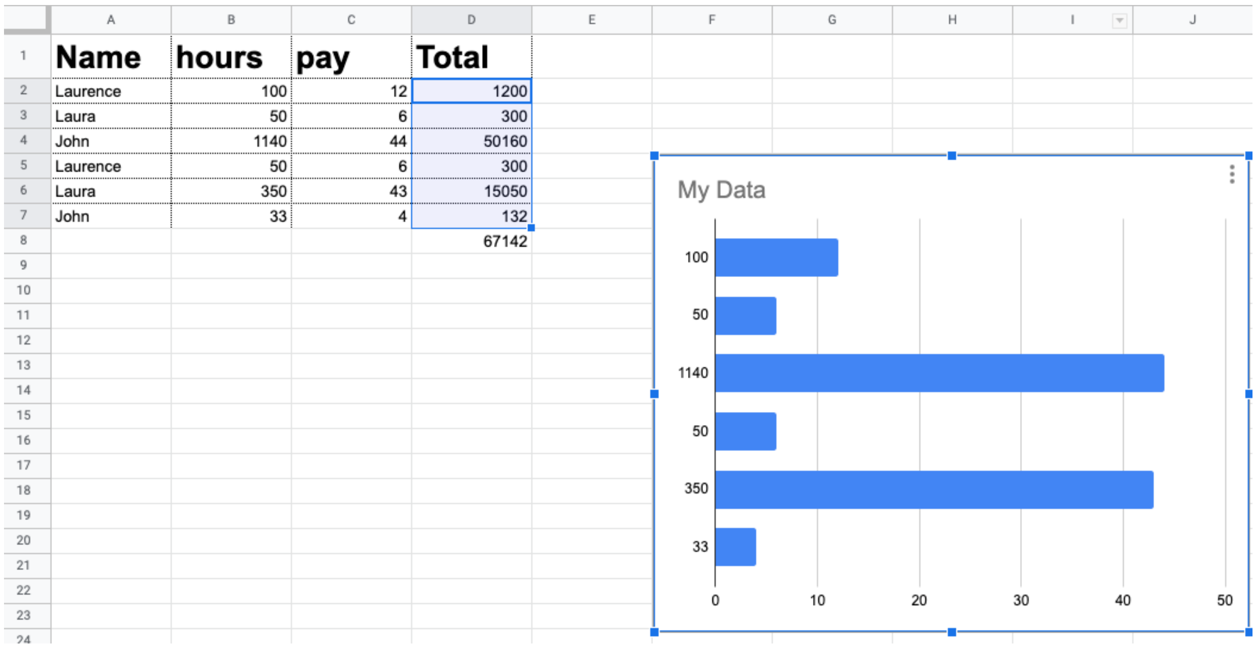
Explanation:

The function formatSheet is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.
The header row is formatted in bold by getting the range of cells in the first row and using the setFontWeight method to set the font weight to "bold".
Borders are added to all cells by getting the range of all cells and using the setBorder method to set all borders to true.
The sum of the values in column 1 is calculated using a for loop and stored in the sum variable.
The sum is written to the bottom of column 1 using the sheet.getRange(data.length + 1, 1).setValue method.

A bar chart is created to visualize the data in column 1 using the Charts service and the sheet.newChart method. The chart options, such as the title and dimensions, are set using the setOption method. The chart is inserted into the sheet using the sheet.insertChart method.
This script can be run in a Google Spreadsheet by opening the script editor (under the "Tools" menu), copying and pasting the code into the editor, and then clicking the "Run" button. It will format the cells in the sheet, calculate the sum of the values in column 1, and insert a chart to visualize the data.

Using Sheet data create a chart update styling

```
function formatSheetData(){
  const sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheets()[1];
  sheet.getRange(1,4,1,1).setValue('Total');
  const data = sheet.getDataRange().getValues();
  const headerRange =
sheet.getRange(1,1,1,data[0].length);
  headerRange.setFontWeight('bold');
  headerRange.setFontSize(20);
  const borderRange =
sheet.getRange(1,1,data.length,data[0].length);

borderRange.setBorder(true,false,false,true,true,true,'
black',SpreadsheetApp.BorderStyle.DOTTED);
  let total = 0;
  for(let i=1;i<data.length;i++){
    let sum = parseInt(data[i][1]) *
parseInt(data[i][2]);
    sheet.getRange(i+1,4,1,1).setValue(sum);
    Logger.log(sum);
    total += sum;
  }
  sheet.getRange(data.length+1,4,1,1).setValue(total);
  const myChart = sheet.newChart()
      .setChartType(Charts.ChartType.BAR)
      .addRange(sheet.getRange('B2:C7'))
      .setOption('title','My Data')
      .setOption('width',500)
      .setOption('height',400)
      .setPosition(5,6,0,0)
      .build();
  sheet.insertChart(myChart)
  Logger.log(total);
}
```



	Name	hours	pay	Total
1	Name	hours	pay	Total
2	Laurence	100	12	1200
3	Laura	50	6	300
4	John	1140	44	50160
5	Laurence	50	6	300
6	Laura	350	43	15050
7	John	33	4	132
8				67142

Apps Script Create Calendar entries from Sheet data

Google Apps Script that automatically creates and updates a Google Calendar event based on data in a Google Sheet:

```
function createCalendarEvent() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();

  for (var i = 1; i < data.length; i++) {
    var startTime = data[i][0];
    var endTime = data[i][1];
    var eventName = data[i][2];
    var calendar =
CalendarApp.getCalendarById(data[i][3]);

    var event = calendar.createEvent(eventName,
startTime, endTime);
  }
}

function updateCalendarEvent() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();

  for (var i = 1; i < data.length; i++) {
    var eventId = data[i][4];
    var event = CalendarApp.getEventById(eventId);

    var startTime = data[i][0];
    var endTime = data[i][1];
    var eventName = data[i][2];
    var calendar =
CalendarApp.getCalendarById(data[i][3]);
```

```
    event.setTitle(eventName);
    event.setStartTime(startTime);
    event.setEndTime(endTime);
    event.setCalendar(calendar);
  }
}
```

Explanation:

The function createCalendarEvent is defined.

The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.

The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.

A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).

For each row, the start time, end time, event name, and calendar ID are stored in the startTime, endTime, eventName, and calendar variables respectively.

A new event is created in the calendar specified by calendar using the calendar.createEvent method, with the start time, end time, and event name specified by startTime, endTime, and eventName respectively.

The function updateCalendarEvent is defined.

The sheet and data variables are set in the same way as in the createCalendarEvent function.

A for loop is used to iterate through the rows in the data array, starting from the second row.

For each row, the event ID is stored in the eventId variable, and the event is retrieved using the CalendarApp.getEventById method.

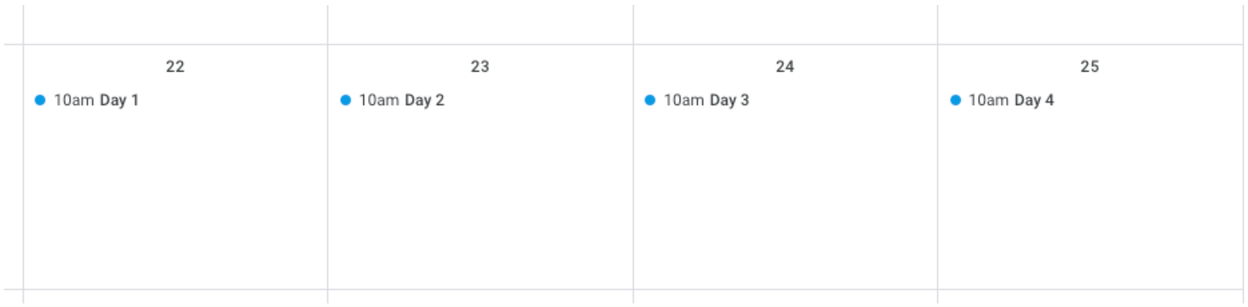
The start time, end time, event name, and calendar ID are stored in the startTime, endTime, eventName, and calendar variables respectively.

The event is updated using the setTitle, setStartTime, setEndTime, and setCalendar methods

Sheet data to create Calendar events

Event	Start	End	Location	Description
Day 1	3/22/2023 10:00:00	3/22/2023 12:00:00	online	Another Day starts 1
Day 2	3/23/2023 10:00:00	3/23/2023 12:00:00	online	Another Day starts 2
Day 3	3/24/2023 10:00:00	3/24/2023 12:00:00	online	Another Day starts 3
Day 4	3/25/2023 10:00:00	3/25/2023 12:00:00	online	Another Day starts 4
Day 5	3/26/2023 10:00:00	3/26/2023 12:00:00	online	Another Day starts 5

```
const ID = '132trziyLsk8qIk';
function makeEvents(){
  const sheet =
SpreadsheetApp.openById(ID).getSheetByName('Cal');
  const data = sheet.getDataRange().getValues();
  const events = data.slice(1);
  const headings = data[0];
  events.forEach((event,index)=>{
    const start = event[1];
    const nameEvent = event[0];
    const end = event[2];
    const loc = event[3];
    const des = event[4];
    const cal =CalendarApp.getDefaultCalendar();
    const eve = cal.createEvent(nameEvent,start,end,{
      location:loc,
      description:des
    });
    Logger.log(cal);
  })
}
```



Calendar Events into Sheet

```
function updateCal(){
  const cal =CalendarApp.getDefaultCalendar();
  const ss = SpreadsheetApp.openById(ID);
  const start = new Date();
  const end = new Date(start.getTime() +
(100*24*60*60*1000));
  const events = cal.getEvents(start,end);
  Logger.log(events);
  const sheet = ss.insertSheet();

sheet.appendRow(['Name','Start','End','Location','ID'])
;
  sheet.setName('New Event List');
  events.forEach(event => {
    const arr =
[event.getTitle(),event.getStartTime(),event.getEndTime
(),event.getLocation(),event.getId()];
    sheet.appendRow(arr);
  })
}
```

Sheet data to populate Calendar

Google Apps Script that copies the data from a Google Sheet to a Google Calendar:

```
function copyToCalendar() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();
  var calendar =
CalendarApp.getCalendarById("your_calendar_id");
```

```
  for (var i = 1; i < data.length; i++) {
    var date = data[i][0];
    var eventName = data[i][1];
    var description = data[i][2];

    var start = new Date(date);
    var end = new Date(date);
    end.setHours(end.getHours() + 1);

    calendar.createEvent(eventName, start, end, { description:
description });
  }
}
```
Explanation:

The function copyToCalendar is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.
The calendar variable is set to the calendar with the specified ID using the CalendarApp.getCalendarById method. Replace "your_calendar_id" with the actual ID of your calendar.

A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).
For each row, the date, event name, and description are stored in the date, eventName, and description variables respectively.
The start variable is set to a new date object created from the date variable.
The end variable is set to a new date object created from the date variable, with the number of hours increased by 1.
A new event is created in the calendar using the calendar.createEvent method, with the event name, start and end times, and description specified by eventName, start, end, and description respectively.

Logs Email Attachments details

Google Apps Script that logs information about all attachments in a Gmail account:

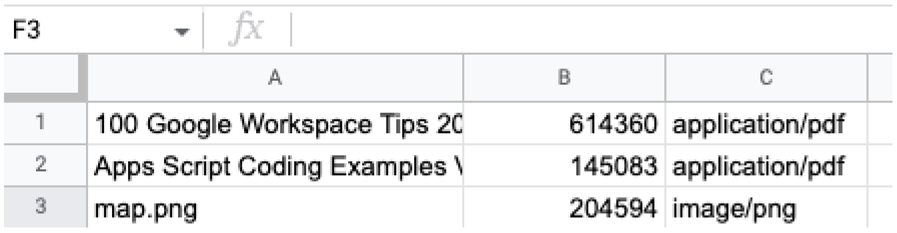
```
function logAttachments() {
  var threads = GmailApp.search("has:attachment");

  for (var i = 0; i < threads.length; i++) {
    var messages = threads[i].getMessages();

    for (var j = 0; j < messages.length; j++) {
      var attachments = messages[j].getAttachments();

      for (var k = 0; k < attachments.length; k++) {
        var attachment = attachments[k];
        var fileName = attachment.getName();
        var size = attachment.getSize();

        Logger.log("Attachment: " + fileName + " - 
Size: " + size + " bytes");
      }
    }
  }
}
```

Explanation:

The function logAttachments is defined.
The threads variable is set to an array of all threads in the Gmail account that have attachments, using the GmailApp.search method with the query "has:attachment".
A for loop is used to iterate through the threads in the threads array.
For each thread, the messages variable is set to an array of all messages in the thread using the threads[i].getMessages method.

Another for loop is used to iterate through the messages in the messages array.
For each message, the attachments variable is set to an array of all attachments in the message using the messages[j].getAttachments method.
Another for loop is used to iterate through the attachments in the attachments array.
For each attachment, the fileName and size variables are set to the name and size of the attachment respectively, using the attachment.getName and attachment.getSize methods.
The information about the attachment is logged to the Google Apps Script log using the Logger.log method.

Get Gmail Attachments Log into Sheet



F3

	A	B	C
1	100 Google Workspace Tips 20	614360	application/pdf
2	Apps Script Coding Examples \	145083	application/pdf
3	map.png	204594	image/png

```
function getAtt(){
  const id = '132trvLa8IvLsk8qIk';
  const threads = GmailApp.search('has:attachment');
  const ss = SpreadsheetApp.openById(id);
  const sheet = ss.insertSheet();
  sheet.setName('Attachments');
  threads.forEach(thread =>{
    const messages = thread.getMessages();
    messages.forEach(message => {
      const attments = message.getAttachments();
      attments.forEach(attachment =>{
        Logger.log(attachment.getName());
        const arr =
[attachment.getName(),attachment.getSize(),attachment.g
etContentType()];
        sheet.appendRow(arr);
      })
    })

  })

}
```

Create folder in drive from Sheet Data

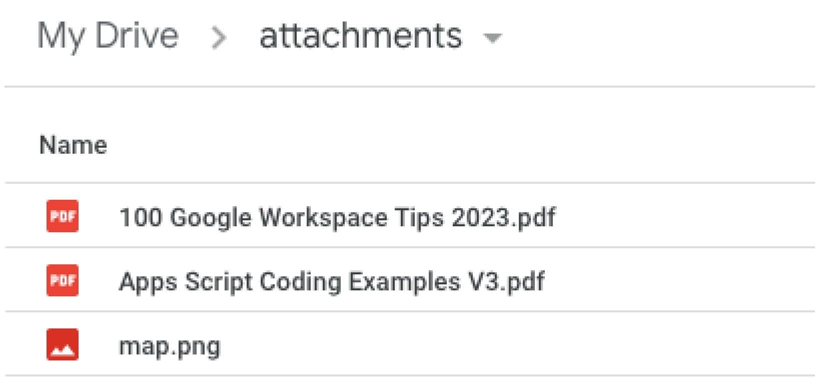
Google Apps Script that creates a new folder in Google Drive for each row in a Google Sheet and stores a PDF file with the contents of a specific column in that folder:

```
function createFoldersAndFiles() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var data = sheet.getDataRange().getValues();
  var parentFolder =
DriveApp.getFolderById("your_parent_folder_id");

  for (var i = 1; i < data.length; i++) {
    var folderName = data[i][0];
    var contents = data[i][1];
    var folder = parentFolder.createFolder(folderName);

    var file = folder.createFile("contents.pdf", contents,
MimeType.PDF);
  }
}
```
Explanation:

The function createFoldersAndFiles is defined.
The sheet variable is set to the active sheet in the current spreadsheet using the SpreadsheetApp.getActiveSheet method.
The data variable is set to the values of all cells in the active sheet using the sheet.getDataRange().getValues method.

The parentFolder variable is set to the folder with the specified ID using the DriveApp.getFolderById method. Replace "your_parent_folder_id" with the actual ID of your parent folder.
A for loop is used to iterate through the rows in the data array, starting from the second row (the first row is assumed to contain headers).
For each row, the folder name and contents are stored in the folderName and contents variables respectively.
A new folder is created in the parent folder using the parentFolder.createFolder method and the folder name is specified by folderName.
A new file with the contents of the contents variable is created in the new folder using the folder.createFile method, with the file name "contents.pdf" and MIME type PDF specified by "contents.pdf" and MimeType.PDF respectively.

Get attachments from Gmail and Copy To Drive


My Drive > attachments

Name
100 Google Workspace Tips 2023.pdf
Apps Script Coding Examples V3.pdf
map.png

```
function makeFolderDrive(){
  const folder = DriveApp.createFolder('attachments');
  const threads = GmailApp.search('has:attachment');
  const attachments = [];
  threads.forEach(thread => {
    thread.getMessages().forEach(message => {
      message.getAttachments().forEach(attachment => {
        const file =
folder.createFile(attachment.copyBlob().setName(attachm
ent.getName()));
        attachments.push(attachment.getName());
      })
    })
  })
  Logger.log(attachments);
}
```