Day 1: Introduction to Google Apps Script

Welcome to your first day! Today is all about understanding what Google Apps Script is and writing your very first line of code.

What is Google Apps Script?

Google Apps Script is a **cloud-based scripting language** for light-weight application development in the Google Workspace platform. It's based on **JavaScript**, so if you know a bit of JS, you're ahead of the game! If not, don't worry—we'll cover the basics. You can use it to automate tasks, connect different Google Apps (like Sheets, Docs, and Gmail), and build custom add-ons.

Think of it as the glue that lets you stick Google's services together and make them do new, cool things. For example, you could write a script that automatically sends an email from Gmail whenever a new row is added to a Google Sheet.

Your First Script: "Hello, World!"

Let's get our hands dirty. The best way to learn is by doing. We'll start from within a Google Sheet.

1. **Create a new Google Sheet**: Go to sheets.new.
2. **Open the Script Editor**: In the menu, go to **Extensions > Apps Script**.
3. **The Script Editor**: A new tab will open with the script editor. You'll see a default function called myFunction.
4. **Write the Code**: Delete the existing code and replace it with this:

   ```
   function sayHello() {
     Logger.log("Hello, World!");
   }
   ```

   - function sayHello(): This declares a function named sayHello. A function is a block of code designed to perform a particular task.
   - Logger.log(): This is a built-in Apps Script command that prints text to the **Execution log**. It's incredibly useful for testing and debugging your code.
5. **Save and Run**:
   - Click the **Save project** icon (looks like a floppy disk 💾).
   - In the dropdown menu next to the "Debug" button, make sure sayHello is selected.
   - Click the **Run** button.
6. **Authorization**: The first time you run a script, Google will ask for your permission. This is normal. Click "Review permissions," select your Google account, click "Advanced," and then "Go to (unsafe)." Finally, click "Allow."

7. **Check the Log**: To see your message, go to **View > Logs** (or press Ctrl+Enter / Cmd+Enter). You should see "Hello, World!" printed there.

Congratulations, you've just run your first Google Apps Script! 🎉

Day 1 Quiz

1. What programming language is Google Apps Script based on?
2. What is the purpose of the Logger.log() command?
3. How do you open the Apps Script editor from a Google Sheet?
4. What must you do the very first time you run a script in a new project?

Day 2: Variables, Data Types, and Basic Operations

Today, we'll learn about the fundamental building blocks of any programming language: how to store and manipulate information.

Variables and Data Types

A **variable** is a container for storing data. You declare a variable using the var, let, or const keywords. For now, we'll stick with let.

Apps Script has several basic data types:

- **String**: Text, enclosed in quotes. Example: let name = "Alice";
- **Number**: Integers or decimals. Example: let age = 30;
- **Boolean**: true or false. Example: let isStudent = false;
- **Array**: A list of items. Example: let fruits = ["Apple", "Banana", "Cherry"];
- **Object**: A collection of key-value pairs. Example: let person = {firstName: "John", lastName: "Doe"};

Coding Example: Working with Data

Let's practice. Replace the code in your script editor with this:

```
function practiceVariables() {
 // 1. Declare variables of different types
 let recipient = "marketing.team@example.com";
 let newSignups = 15;
 let campaignIsActive = true;
 let quarterlyGoals = [50, 75, 100, 125];

 // 2. Perform a simple operation
```

```
  let subjectLine = "Update: " + newSignups + " new signups today!";

  // 3. Log the results to check your work
  Logger.log(recipient);
  Logger.log("New Signups: " + newSignups);
  Logger.log("Is the campaign active? " + campaignIsActive);
  Logger.log("Second quarter goal: " + quarterlyGoals[1]); // Arrays are 0-indexed, so [1] is the second
item.
  Logger.log(subjectLine);
}
```

Save and **Run** the practiceVariables function. Check the execution log to see the output. You'll notice how we combined a string and a number ("Update: " + newSignups)—this is called **concatenation**.

Day 2 Quiz

1. What is the difference between a String and a Number data type?
2. How would you declare a variable named userName and assign it the value "test_user"?
3. In the array let colors = ["Red", "Green", "Blue"];, how would you access the value "Green"?
4. What will Logger.log("Score: " + 5 + 5); output?

Day 3: Interacting with Google Sheets

This is where the magic begins! Today, we'll learn how to make our scripts talk to Google Sheets.

The SpreadsheetApp Service

To work with Google Sheets, we use the SpreadsheetApp service, which is a built-in object that gives you access to all the necessary methods.

Here are the most important ones to start:

- SpreadsheetApp.getActiveSpreadsheet(): Gets the spreadsheet file the script is currently attached to.
- spreadsheet.getSheetByName("Sheet1"): Gets a specific sheet (tab) by its name.
- sheet.getRange("A1"): Gets a specific cell or range of cells.
- range.getValue(): Gets the value from a single cell (e.g., "A1").
- range.getValues(): Gets the values from a range of cells as a 2D array.
- range.setValue("Hello"): Sets the value of a single cell.

Coding Example: Read and Write to a Sheet

1. In your Google Sheet, type "Status" in cell **A1** and "Not Started" in cell **B1**.
2. Go to your script editor and use this code:

```
function updateSheetStatus() {
 // 1. Get the active spreadsheet and the specific sheet.
 const ss = SpreadsheetApp.getActiveSpreadsheet();
 const sheet = ss.getSheetByName("Sheet1"); // Make sure your sheet name is "Sheet1"

 // 2. Get the range B1 and read its current value.
 const statusCell = sheet.getRange("B1");
 const currentStatus = statusCell.getValue();
 Logger.log("The current status is: " + currentStatus);

 // 3. Update the value of cell B1.
 statusCell.setValue("In Progress");
 Logger.log("The status has been updated!");

 // 4. Write a new label and value.
 sheet.getRange("A2").setValue("Completion Date:");
 sheet.getRange("B2").setValue(new Date()); // new Date() inserts the current date and time.
}
```

Save and **Run** the updateSheetStatus function. Now, look at your Google Sheet. You'll see that cell B1 now says "In Progress" and cells A2/B2 have been populated!

Day 3 Quiz

1. What method is used to get the currently open Google Sheet file?
2. What's the difference between getValue() and getValues()?
3. Write the line of code to get cell C5 from a sheet named "Data".
4. What does the setValue() method do?

Day 4: Conditional Logic with 'If' Statements

Conditional logic allows your script to make decisions and execute different actions based on specific conditions.

Making Decisions: if, else if, else

The if statement is the core of decision-making in code. Its structure is simple:

```
if (condition) {
  // Code to run if the condition is true
} else {
  // Code to run if the condition is false
}
```

The **condition** is an expression that evaluates to a boolean (true or false). You can use comparison operators like:

- == (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

Coding Example: Check a Value in a Sheet

Let's create a script that checks the value of a cell and writes a comment based on it.

1. In your Google Sheet, type 55 into cell **A1**.
2. Use this code in the script editor:

```
function checkScore() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheetByName("Sheet1");
  const scoreCell = sheet.getRange("A1");
  const score = scoreCell.getValue();

  const resultCell = sheet.getRange("B1");

  if (score > 90) {
    resultCell.setValue("Excellent");
  } else if (score > 70) {
    resultCell.setValue("Good");
  } else if (score >= 50) {
    resultCell.setValue("Pass");
  } else {
    resultCell.setValue("Fail");
  }
}
```

Save and **Run** checkScore. Cell B1 should now say "Pass". Try changing the number in A1 to 95, 75, or 40 and re-running the script to see how the output in B1 changes.

Day 4 Quiz

1. What is the purpose of an if statement?
2. What is the difference between == and =?
3. How do you execute code only when multiple conditions are ALL true? (Hint: Use &&)
4. What does the else block do?

Day 5: Automation with Loops

Loops are used to repeat a block of code multiple times. This is essential for processing multiple rows of data in a spreadsheet.

---

for Loops

The most common type of loop is the for loop. It's perfect for iterating over an array, like the data you get from a range in Google Sheets.

The basic syntax is: for (initializer; condition; increment) { ... }

- **Initializer**: Runs once at the very beginning (e.g., let i = 0;).
- **Condition**: Checked before each iteration. The loop continues as long as this is true (e.g., i < 10;).
- **Increment**: Runs at the end of each iteration (e.g., i++, which means i = i + 1).

Coding Example: Process Multiple Rows

1. Set up your Google Sheet like this:
    - **A1**: "Status"
    - **B1**: "Priority"
    - **A2**: "Pending"
    - **A3**: "Done"
    - **A4**: "Pending"
    - **A5**: "In Progress"

2. Use this code to loop through the "Status" column and set a "Priority" based on the status.

```
function setPriorities() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheetByName("Sheet1");

  // Get all the data from A2 to the last row with content.
  const dataRange = sheet.getRange("A2:A" + sheet.getLastRow());
  const statuses = dataRange.getValues(); // This is a 2D array: [["Pending"], ["Done"], ...]

  // Loop through each row of the statuses array.
  for (let i = 0; i < statuses.length; i++) {
    const currentStatus = statuses[i][0]; // Get the value from the current row.
    let priority = ""; // Default priority.

    if (currentStatus === "Pending") {
      priority = "High";
    } else if (currentStatus === "In Progress") {
      priority = "Medium";
    } else if (currentStatus === "Done") {
      priority = "Low";
    }

    // Set the priority value in the corresponding cell in column B.
    // i + 2 because our data starts at row 2.
    sheet.getRange(i + 2, 2).setValue(priority); // getRange(row, column)
  }
}
```

Save and **Run** setPriorities. Your sheet's "Priority" column will now be filled in automatically!

Day 5 Quiz

1. Why are loops useful when working with spreadsheets?
2. In a for loop, what does the i++ part typically do?
3. If getValues() returns a 2D array for a single column, how do you access the actual value inside the loop?
4. What does sheet.getLastRow() return?

Day 6: Creating Your Own Functions

You've been using built-in functions like Logger.log() and custom ones like setPriorities(). Today, you'll learn to make your functions more powerful and reusable with arguments and return values.

Reusable Blocks of Code

A function can accept **arguments** (inputs) and **return** a value (output). This makes them incredibly flexible.

```
function calculateVat(price, vatRate) {
  let vatAmount = price * vatRate;
  return vatAmount; // Send this value back as the output.
}

function useTheFunction() {
  let productPrice = 100;
  let taxRate = 0.13; // 13%

  // Call the function and store its output in a variable.
  let taxToPay = calculateVat(productPrice, taxRate);

  Logger.log("The VAT to pay is: $" + taxToPay); // Logs "The VAT to pay is: $13"
}
```

Custom Functions in Google Sheets

One of the coolest features of Apps Script is creating custom functions you can use directly in your sheet, just like =SUM() or =AVERAGE().

- The function must return a value.
- The function name becomes the formula name in the sheet.

Coding Example: A Custom Sheet Function

Let's create a custom function to calculate the final price after tax.

```
/**
 * Calculates the final price including sales tax.
 *
 * @param {number} price The initial price of the item.
 * @param {number} taxRate The tax rate as a decimal (e.g., 0.13 for 13%).
 * @return The final price with tax.
 * @customfunction
 */
```

```
function ADD_TAX(price, taxRate) {
  if (typeof price !== 'number' || typeof taxRate !== 'number') {
    return "Please enter valid numbers.";
  }
  const finalPrice = price * (1 + taxRate);
  return finalPrice;
}
```

Save the script. Now go back to your Google Sheet. In any cell, type the formula =ADD_TAX(100, 0.13). The cell will display 113. You've made your own Excel-style function! The text inside the /** ... */ is a special comment that provides auto-complete information within the sheet.

Day 6 Quiz

1. What is the purpose of the return keyword in a function?
2. What is an "argument" in the context of a function?
3. What makes a function a "custom function" that you can use in a Sheet? (Hint: See the comments in the code).
4. Write a simple function called DOUBLE that takes one number as an argument and returns that number multiplied by 2.

Day 7: Working with Google Docs and Gmail

Apps Script isn't just for Sheets! You can control many other Google services. Today we'll automate creating a Google Doc and sending an email notification.

Interacting with Other Services

Just like SpreadsheetApp, there are other global objects for different services:

- DocumentApp: For Google Docs.
- GmailApp: For Gmail.
- CalendarApp: For Google Calendar.
- And many more!

Coding Example: Create a Report and Email It

This script will take data from our sheet, create a summary in a new Google Doc, and then email a link to that doc.

1. Make sure your sheet from Day 5 is set up (with Status and Priority columns).
2. Use this code in the script editor:

```
function createAndEmailReport() {
 // --- Part 1: Get data from the Sheet ---
 const ss = SpreadsheetApp.getActiveSpreadsheet();
 const sheet = ss.getSheetByName("Sheet1");
 const data = sheet.getDataRange().getValues(); // Get ALL data

 // --- Part 2: Create a new Google Doc ---
 const doc = DocumentApp.create('Task Summary Report');
 const body = doc.getBody();
 body.appendParagraph('Daily Task Report'); // Add a title

 // Loop through the sheet data (skip the header row, so i=1)
 for (let i = 1; i < data.length; i++) {
   let taskStatus = data[i][0]; // Column A
   let taskPriority = data[i][1]; // Column B
   body.appendListItem(`Status: ${taskStatus}, Priority: ${taskPriority}`);
 }
 doc.saveAndClose(); // Important!

 // --- Part 3: Send an email with a link to the Doc ---
 const docUrl = doc.getUrl();
 const recipient = "youremail@example.com"; // !! CHANGE THIS TO YOUR EMAIL !!
 const subject = "Daily Task Report is Ready";
 const message = "Here is the link to today's report: " + docUrl;

 GmailApp.sendEmail(recipient, subject, message);

 Logger.log("Report created and email sent!");
}
```

Save and **Run** createAndEmailReport. You'll need to authorize the script again for Docs and Gmail permissions. After it runs, check your Google Drive for the new Doc and your Gmail for the notification!

Day 7 Quiz

1. What service would you use to create a new Google Doc programmatically?
2. What does doc.getBody() do?

3. What are the three essential arguments for the GmailApp.sendEmail() method?
4. How can you get all the data from a sheet in a single command?

Day 8: Triggers - Automating Your Scripts

Triggers automatically run a function when a certain event occurs, like opening a document or at a specific time each day. This is true automation!

Simple Triggers

Simple triggers are special, reserved function names that run automatically. You don't need to set them up anywhere else.

- onOpen(e): Runs when a user opens the spreadsheet, doc, or form. It's often used to add a custom menu.
- onEdit(e): Runs when a user changes the value of any cell in a spreadsheet.

Installable Triggers

Installable triggers offer more flexibility. You can set them up to run:

- On a time-driven schedule (e.g., every day between 8-9 AM).
- In response to events like a form submission.

Coding Example: onOpen() and Time-Driven Triggers

Part 1: A Custom Menu
Add this onOpen() function to your script. It will create a new menu in your spreadsheet.

```
function onOpen() {
  SpreadsheetApp.getUi() // Get the user interface environment for the spreadsheet.
      .createMenu('Automation')
      .addItem('Create Report', 'createAndEmailReport') // Menu item text and function to run
      .addToUi();
}
```

Save the script and **reload your Google Sheet**. You should now see a new menu called "Automation" next to "Help." Clicking "Create Report" will run the function we made on Day 7.

Part 2: A Time-Driven Trigger
Let's make the report run automatically every morning.

1. In the script editor, click the **Triggers** icon (looks like an alarm clock ⏰) on the left sidebar.
2. Click **+ Add Trigger** in the bottom right.
3. Set it up as follows:

   - **Choose which function to run**: createAndEmailReport
   - **Select event source**: Time-driven
   - **Select type of time-based trigger**: Day timer
   - **Select time of day**: 8am to 9am (or your preference)
4. Click **Save**.

Now, your createAndEmailReport function will run automatically every morning without you needing to do anything.

Day 8 Quiz

1. What is the difference between a simple trigger and an installable trigger?
2. What is the name of the function that runs automatically when you open a spreadsheet?
3. What is the purpose of the SpreadsheetApp.getUi() method?
4. Where in the Apps Script editor do you go to set up a time-based trigger?

Day 9: Creating a User Interface (UI)

While running scripts from the editor is fine for development, you often want to give users an easier way to interact with your code.

Alerts, Prompts, and Menus

We already created a custom menu on Day 8. The SpreadsheetApp.getUi() object also lets us create simple dialog boxes.

- ui.alert('Message'): A simple dialog with an "OK" button.
- ui.prompt('Question', ui.ButtonSet.OK_CANCEL): A dialog that asks the user for text input.

These are great for confirming an action or getting a small piece of information from the user.

Coding Example: A UI-Driven Workflow

Let's create a script that asks the user for a status, then highlights all rows with that status. We'll add it to our custom menu.

1. First, add this new function to your script:

```
function highlightRowsByStatus() {
  const ui = SpreadsheetApp.getUi();

  // 1. Ask the user for input.
  const result = ui.prompt(
    'Highlight Status',
    'Enter the status to highlight (e.g., Pending, Done):',
    ui.ButtonSet.OK_CANCEL);

 // 2. Check if the user clicked OK.
 const button = result.getSelectedButton();
 const inputText = result.getResponseText();

 if (button == ui.Button.OK) {
   // 3. If they clicked OK, proceed with the script.
   const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Sheet1");
   const dataRange = sheet.getDataRange();
   const values = dataRange.getValues();

   // Loop through all rows, skipping the header.
   for (let i = 1; i < values.length; i++) {
     let status = values[i][0]; // Status is in the first column (index 0).

     if (status.toLowerCase() === inputText.toLowerCase()) { // Case-insensitive comparison
       // Highlight the entire row.
       sheet.getRange(i + 1, 1, 1, sheet.getLastColumn()).setBackground("#f9f68a"); // A yellow
color
     }
   }
   ui.alert('Highlighting complete for status: ' + inputText);
 } else {
   ui.alert('Action canceled.');
 }
}
```

2. Next, **update your onOpen function** to include this new script in the menu:

```
function onOpen() {
 SpreadsheetApp.getUi()
    .createMenu('Automation')
    .addItem('Create Report', 'createAndEmailReport')
    .addSeparator() // Adds a line to separate menu items
    .addItem('Highlight by Status', 'highlightRowsByStatus')
    .addToUi();
}
```

Save and **reload your sheet**. Go to **Automation > Highlight by Status**, enter "Pending" in the prompt, and watch the corresponding rows turn yellow!

Day 9 Quiz

1. What method do you use to show a simple pop-up message to the user?
2. How do you get the text a user entered into a ui.prompt?
3. Why is it important to check which button the user clicked in a prompt dialog?
4. In the getRange(row, column, numRows, numColumns) method, what do the four parameters represent?

Day 10: Project Day - Putting It All Together

It's time to combine everything you've learned into a final, practical project.

Project: Invoice Generator

The Goal: Create a script that takes invoice data from one sheet ("InvoiceData"), generates a formatted invoice in a Google Doc from a template, and saves a PDF copy of that invoice to a specific Google Drive folder.

Step 1: The Setup

1. **Google Sheet**:
   - Rename a sheet to InvoiceData.
   - Set it up with these headers in row 1: Client Name, Item, Quantity, Unit Price, Invoice Number.
   - Add some sample data in row 2.
2. **Google Drive Folder**:
   - Create a new folder in your Google Drive and name it Invoices.
   - Open the folder and copy its ID from the URL bar. It's the long string of letters and numbers at the end. (e.g., .../folders/1a2b3c...)
3. **Google Doc Template**:
   - Create a new Google Doc. This will be your template.
   - Write your invoice layout using placeholder text, like:
     - **Invoice for:** {{Client Name}}
     - **Invoice #:** {{Invoice Number}}
     - **Line Item:** {{Item}}
     - **Amount Due:** {{Total}}
   - These {{...}} placeholders are what our script will find and replace.

Step 2: The Code

```
// --- CONFIGURATION ---
const INVOICE_TEMPLATE_ID = "YOUR_TEMPLATE_DOC_ID_HERE"; // Get this from the template
Doc's URL
const INVOICES_FOLDER_ID = "YOUR_DRIVE_FOLDER_ID_HERE"; // Get this from the Drive
folder's URL

function createInvoice() {
  // 1. Get the data from the active row in the Sheet.
  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("InvoiceData");
  const activeRow = sheet.getActiveRange().getRow();

  // Make sure we're not on the header row.
  if (activeRow < 2) {
    SpreadsheetApp.getUi().alert("Please select a row with invoice data.");
    return;
  }
  const data = sheet.getRange(activeRow, 1, 1, 5).getValues()[0];
  const clientName = data[0];
  const item = data[1];
  const quantity = data[2];
  const unitPrice = data[3];
  const invoiceNum = data[4];
  const total = quantity * unitPrice;

  // 2. Make a copy of the template document.
  const templateFile = DriveApp.getFileById(INVOICE_TEMPLATE_ID);
  const destinationFolder = DriveApp.getFolderById(INVOICES_FOLDER_ID);
  const newFileName = `Invoice_${invoiceNum}_${clientName}.pdf`;

  const newInvoiceCopy = templateFile.makeCopy(`${invoiceNum}_${clientName}`,
destinationFolder);
  const doc = DocumentApp.openById(newInvoiceCopy.getId());
  const body = doc.getBody();

  // 3. Replace the placeholder text with our data.
  body.replaceText("{{Client Name}}", clientName);
  body.replaceText("{{Invoice Number}}", invoiceNum);
  body.replaceText("{{Item}}", item);
  body.replaceText("{{Quantity}}", quantity);
  body.replaceText("{{Unit Price}}", unitPrice);
  body.replaceText("{{Total}}", total.toFixed(2)); // .toFixed(2) formats it as currency.

  // 4. Save the doc, create the PDF, and clean up.
  doc.saveAndClose();
  const pdf = newInvoiceCopy.getAs('application/pdf');
  pdf.setName(newFileName);
  destinationFolder.createFile(pdf); // Create the PDF in the folder
  newInvoiceCopy.setTrashed(true); // Delete the temporary Google Doc copy

  SpreadsheetApp.getUi().alert("Invoice created successfully!");
}

// Add a menu to run the script easily.
function onOpen() {
  SpreadsheetApp.getUi()
      .createMenu('Invoicing')
      .addItem('Generate Invoice from Selected Row', 'createInvoice')
      .addToUi();
}
```

Step 3: Run and Test

1. Paste the code into your script editor.
2. **Crucially**, update the INVOICE_TEMPLATE_ID and INVOICES_FOLDER_ID variables with your own IDs.
3. Save the script and reload your sheet.
4. Click on any cell in row 2 (your data row).
5. Go to **Invoicing > Generate Invoice from Selected Row**.
6. Authorize the script (for Drive and Docs).
7. Check your "Invoices" folder in Google Drive. You should see a brand new PDF invoice!

Final Quiz

1. How do you get a specific file or folder from Google Drive using its ID?
2. What does the body.replaceText() method do in DocumentApp?
3. Why is it a good practice to delete the temporary Google Doc copy (newInvoiceCopy) after creating the PDF?
4. How could you extend this project? (e.g., email the PDF, log the creation date back in the sheet, etc.)

Congratulations on completing your 10-day journey! You now have a solid foundation in Google Apps Script. The possibilities for automation are nearly endless. Keep experimenting! 🚀