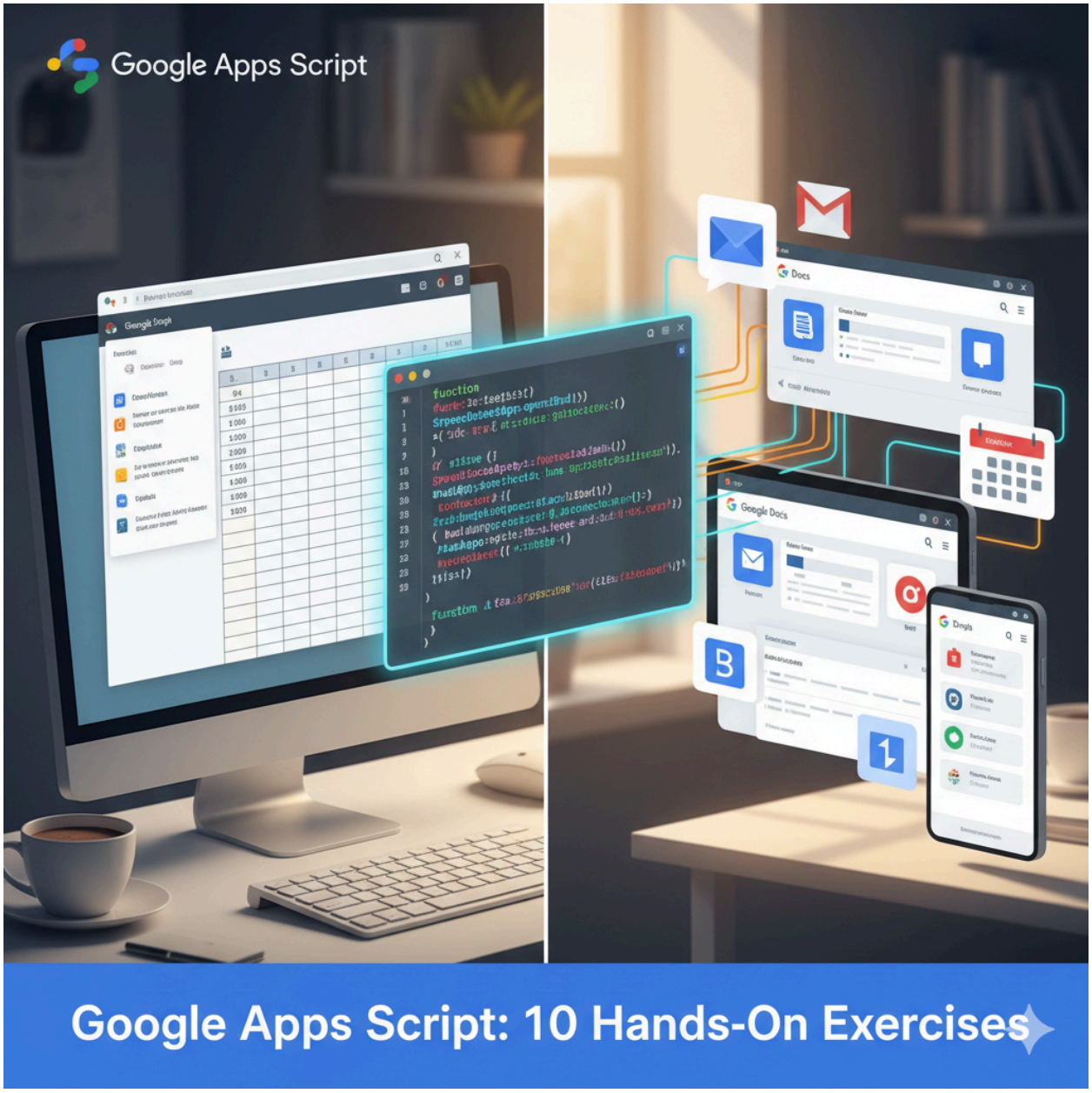
Get Ready to Automate: An Introduction to Your Google Apps Script Exercises



How to Use These Exercises

1. Open any Google Sheet.
2. Go to **Extensions** > **Apps Script**.
3. This will open the Apps Script editor. Clear any default code.

4. Copy and paste the code for each exercise into the editor, save it (Ctrl+S or Cmd+S), and run it by clicking the "Run" button (▶️).
5. You may need to grant permissions the first time you run a script that interacts with your Google data (like Sheets, Docs, or Mail).

Exercise 1: Hello, World! 👋

Objective: Learn the most basic function of Apps Script: creating a function and logging a message to the execution log.

Instructions: Write a script that logs the message "Hello, World!" to the console.

Solution:

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

Explanation:

- `function helloWorld() { ... }` defines a new function named `helloWorld`.
- `Logger.log()` is a built-in command that prints text to the Apps Script execution log. You can view the log by clicking **Execution log** after running the script.

Exercise 2: Send a Simple Email 📧

Objective: Use the `MailApp` service to send an email from your Google account.

Instructions: Write a script that sends an email to your own email address with the subject "Test Email" and the body "This email was sent from Google Apps Script."

Solution:

```
function sendEmail() {
  var userEmail = Session.getActiveUser().getEmail(); // Gets your email address
  var subject = "Test Email";
  var body = "This email was sent from Google Apps Script.";
```

```
  MailApp.sendEmail(userEmail, subject, body);
  Logger.log("Email sent to " + userEmail);
}
```

Explanation:

- `Session.getActiveUser().getEmail()` securely gets the email address of the person running the script.
- `MailApp.sendEmail(recipient, subject, body)` is the core command from the Mail service. It takes the recipient's email, the subject line, and the email body as arguments and sends the email.

Exercise 3: Add Data to a Google Sheet 📝

Objective: Interact with Google Sheets by adding a new row of data.

Instructions: Write a script that adds a new row to the active spreadsheet. The row should contain the current date and the text "New Entry".

Solution:

```
function addDataToSheet() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var currentDate = new Date();
  sheet.appendRow([currentDate, "New Entry"]);
}
```

Explanation:

- `SpreadsheetApp.getActiveSpreadsheet()` gets the Google Sheet file that the script is bound to.
- `.getActiveSheet()` gets the currently active tab (sheet) within that file.
- `new Date()` creates a JavaScript Date object for the current time.
- `sheet.appendRow([...])` adds a new row to the bottom of the sheet. The data must be an array, where each element corresponds to a cell in the new row.

Exercise 4: Read Data from a Google Sheet 🧐

Objective: Read all the data from a sheet and log it to the console.

Instructions: Write a script that gets all the data from the active sheet and logs each row.

Solution:

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

  for (var i = 0; i < data.length; i++) {
    Logger.log("Row " + (i + 1) + ": " + data[i].join(", "));
  }
}
```

Explanation:

- `sheet.getDataRange()` selects all the cells in the sheet that contain data.
- `.getValues()` returns the data as a two-dimensional array (an array of rows, where each row is an array of cell values).
- The `for` loop iterates through the outer array (the rows), and `Logger.log()` prints the content of each inner array (the cells).

Exercise 5: Create a Custom Menu ⚙️

Objective: Add a custom menu to the Google Sheet UI that can run your functions.

Instructions: Create a script that adds a menu named "My Tools" to the Google Sheet UI when the file is opened. This menu should have one item, "Add Data", which runs the `addDataToSheet` function from Exercise 3.

Solution:

```
function onOpen() {
  SpreadsheetApp.getUi()
```

```
    .createMenu('My Tools')
    .addItem('Add Data', 'addDataToSheet')
    .addToUi();
}

function addDataToSheet() {
  // Use the function from Exercise 3
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var currentDate = new Date();
  sheet.appendRow([currentDate, "New Entry"]);
}
```

Explanation:

- `onOpen()` is a **simple trigger**. This special function name tells Apps Script to run the code automatically every time the spreadsheet is opened.
- `SpreadsheetApp.getUi()` gets the user interface environment for the spreadsheet.
- `.createMenu('My Tools')` starts building a new menu.
- `.addItem('Add Data', 'addDataToSheet')` adds an option to the menu. The first argument is the visible text, and the second is the name of the function to run when clicked.
- `.addToUi()` completes the process and adds the menu to the sheet's interface.

Exercise 6: Create a Google Doc from Sheet Data 📄

Objective: Combine two services (`SpreadsheetApp` and `DocumentApp`) to generate a document.

Instructions: Write a script that takes the value from cell A1 in the active sheet and uses it as the title for a new Google Doc. The script should then add the value from cell B1 as the first paragraph.

Solution:

```
function createDocFromSheet() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
```

```
  // Get data from cells A1 and B1
  var docTitle = sheet.getRange("A1").getValue();
  var docBodyText = sheet.getRange("B1").getValue();

  // Create a new Google Doc
  var doc = DocumentApp.create(docTitle);

  // Get the body of the doc and add the text
  var body = doc.getBody();
  body.appendParagraph(docBodyText);

  Logger.log("Document created: " + doc.getUrl());
}
```

Explanation:

- `sheet.getRange("A1").getValue()` gets the data from a specific cell.
- `DocumentApp.create(docTitle)` creates a new Google Doc file in your Google Drive with the specified title.
- `doc.getBody()` gets the main body section of the document.
- `body.appendParagraph()` adds a new paragraph with the text from cell B1.
- `doc.getUrl()` provides the URL to the newly created document, which we log for easy access.

Exercise 7: Show a Custom Alert 🚨

Objective: Create a simple pop-up alert in the user interface.

Instructions: Write a script that shows an alert box with the message "Task Complete!" and an "OK" button.

Solution:

```
function showAlert() {
```

```
  SpreadsheetApp.getUi().alert("Task Complete!");
}
```

Explanation:

- This is a simpler use of the UI service. The `.alert()` method displays a standard browser alert dialog to the user.

Exercise 8: Find and Replace Text in a Doc 🔍

Objective: Programmatically edit the content of an existing Google Doc.

Instructions: Create a function that searches the body of a specific Google Doc for all instances of the word "old" and replaces them with "new". *Note: You'll need to create a Google Doc with the word "old" in it and replace `'YOUR_DOCUMENT_ID'` with its actual ID from the URL.*

Solution:

```
function findAndReplace() {
  // Get the document ID from its URL
  // e.g., docs.google.com/document/d/DOCUMENT_ID/edit
  var docId = 'YOUR_DOCUMENT_ID';

  var doc = DocumentApp.openById(docId);
  var body = doc.getBody();

  var replaced = body.replaceText("old", "new");
  Logger.log("Replacements made: " + replaced.getNumReplacements());
}
```

Explanation:

- `DocumentApp.openById(docId)` opens a specific Google Doc using its unique ID.
- `body.replaceText(searchPattern, replacement)` is a powerful method that

finds all occurrences of a string and replaces them. It can also use more complex search patterns (regular expressions).

Exercise 9: Create a Calendar Event 📅

Objective: Use the `CalendarApp` service to create an event in your primary Google Calendar.

Instructions: Write a script that creates a one-hour event on your calendar for tomorrow at 10 AM called "Project Meeting".

Solution:

```
function createCalendarEvent() {
  var calendar = CalendarApp.getDefaultCalendar();

  // Set the start and end times for tomorrow
  var startTime = new Date();
  startTime.setDate(startTime.getDate() + 1); // Go to tomorrow
  startTime.setHours(10, 0, 0); // Set time to 10:00:00 AM

  var endTime = new Date(startTime.getTime()); // Copy start time
  endTime.setHours(endTime.getHours() + 1); // Add one hour

  // Create the event
  var event = calendar.createEvent("Project Meeting", startTime, endTime);
  Logger.log("Event created with ID: " + event.getId());
}
```

Explanation:

- `CalendarApp.getDefaultCalendar()` gets your main calendar.
- We use JavaScript's `Date` object to define the start and end times for the event.
- `calendar.createEvent(title, startTime, endTime)` creates the event on

your calendar with the specified details.

Exercise 10: Create a Simple Web App 🌐

Objective: Deploy a script as a basic web app that anyone can visit.

Instructions: Write a script that, when visited via its web app URL, displays the text "Welcome to my web app!".

Solution:

```
function doGet() {
  return HtmlService.createHtmlOutput("<h1>Welcome to my web app!</h1>");
}
```

Explanation:

- `doGet()` is another special function. It runs whenever someone visits the script's deployed web app URL using a GET request (the standard way a browser visits a page).
- `HtmlService.createHtmlOutput()` creates an HTML object that can be served to a web browser. The function must return this type of object.
- **To deploy it**:
    1. Save the script.
    2. Click the **Deploy** button > **New deployment**.
    3. Select **Web app** as the type.
    4. Under "Who has access," choose **Anyone**.
    5. Click **Deploy**.
    6. Copy the provided Web app URL and paste it into your browser to see the result.