Google Apps Script Starter Projects

Google Apps Script Exercises

Explore what you can do with Google Apps Script in Workspace.

Google Apps Script use cases, providing both an outline of steps and key code snippets to guide you through the completion of each exercise.

1. Sync Google Sheets with External API Data

Objective: Write a script that fetches data from an external API and updates a Google Sheet with the fetched data. This exercise focuses on integrating external data sources.

Key Steps:

1. Setup Google Sheets for Data: Prepare a sheet with headers matching the data structure you expect from the API.

2. Fetch Data from the API: Use UrlFetchApp to call the API and parse the JSON response.
3. Update the Sheet: Loop through the API response and populate the sheet with new data.

Snippet:

```
function updateSheetWithAPIData() {
 var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("E
xternalData");
 var url = 'https://api.example.com/data'; // Replace
with your API endpoint
 var response = UrlFetchApp.fetch(url);
 var json = JSON.parse(response.getContentText());
 var data = json.map(item => [item.id, item.name,
item.value]); // Adjust based on your API response
structure

 sheet.getRange(2, 1, data.length,
data[0].length).setValues(data); // Assuming row 1 has
headers
}
```

2. Generate Google Calendar Events from Sheets Data

Objective: Create a script that reads event data from a Google Sheet and creates corresponding events in Google Calendar.

Key Steps:

1. Prepare Event Data in Sheets: Include event name, start time, end time, and description.
2. Read Data from Sheets: Use SpreadsheetApp to access the event data.
3. Create Calendar Events: For each row, create a new event in a specific Google Calendar using CalendarApp.

Snippet:

```
function createCalendarEvents() {
 var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("E
vents");
 var rows = sheet.getDataRange().getValues();

 var calendar = CalendarApp.getDefaultCalendar(); // or
getCalendarById('your-calendar-id')
 rows.forEach(function(row, index) {
 if (index === 0) return; // Skip header
 var title = row[0];
```

```
 var startTime = new Date(row[1]);
 var endTime = new Date(row[2]);
 var description = row[3];
 calendar.createEvent(title, startTime, endTime,
{description: description});
 });
}
```

3. Automated Invoice Generator

Objective: Automate the creation of invoices in Google Docs from order data stored in a Google Sheet.

Key Steps:

1. Prepare Order Data in Sheets: Structure a sheet with order details, including customer information and order items.
2. Generate Invoices: For each order, create a new Google Doc from a template and populate it with the order details.
3. Save Invoices to Drive: Optionally, organize the generated invoices in a specific Google Drive folder.

Snippet:

```
function generateInvoices() {
```

```
 var ordersSheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("O
rders");
 var orders = ordersSheet.getDataRange().getValues();
 var templateId = 'your-template-doc-id';

 orders.forEach(function(order, index) {
 if (index === 0) return; // Skip header
 var docCopy =
DriveApp.getFileById(templateId).makeCopy(order[0] + "
Invoice");
 var doc = DocumentApp.openById(docCopy.getId());
 var body = doc.getBody();

 body.replaceText('{{CustomerName}}', order[1]);
 body.replaceText('{{OrderDate}}', order[2]);
 // Replace other placeholders with actual order
details

 doc.saveAndClose();
 });
}
```

4. Custom Data Validation and Cleanup Tool

Objective: Develop a script that checks for and corrects common data entry errors in a Google Sheet (e.g., formatting issues, duplicates).

Key Steps:

1. Identify Common Data Issues: Define what constitutes an error in your dataset (e.g., incorrect formats, duplicates).
2. Scan the Sheet for Errors: Loop through the data, identifying any cells that violate your rules.
3. Correct Errors: Automatically correct errors where possible, or flag them for manual review.

Snippet:

```
function validateAndCleanData() {
 var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("D
ata");
 var range = sheet.getDataRange();
 var data = range.getValues();

 data.forEach(function(row, rowIndex) {
 row.forEach(function(cell, cellIndex) {
 // Example: Check for duplicate entries
```

```
 if (rowIndex > 0 && sheet.getRange(1, cellIndex + 1,
rowIndex, 1).getValues().flat().includes(cell)) {
 sheet.getRange(rowIndex + 1, cellIndex +
1).setBackground('yellow'); // Flag duplicates with
yellow background
 }
 // Add more validation checks as needed
 });
 });
}
```

5. Sync Contacts Between Google Sheets and Google Contacts

Objective: Synchronize contact information between a Google Sheet and Google Contacts, ensuring both are up-to-date.

Key Steps:

1. Read Contacts from Google Sheets: Access a sheet containing contact names, emails, and phone numbers.
2. Fetch Existing Contacts from Google Contacts: Use ContactsApp to get the current list of contacts.
3. Update and Create Contacts: Compare sheet data with existing contacts, update any existing ones, and create new ones as needed.

Snippet:

```
function syncContacts() {
 var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("C
ontacts");
 var data = sheet.getDataRange().getValues();
 var contacts = ContactsApp.getContacts();

 data.forEach(function(row, index) {
 if (index === 0) return; // Skip header
 var contact = contacts.find(contact =>
contact.getPrimaryEmail() === row[1]);
 if (contact) {
 // Update existing contact
 contact.setGivenName(row[0]);
 contact.setPrimaryEmail(row[1]);
 contact.setMobilePhone(row[2]);
 } else {
 // Create new contact
 var newContact = ContactsApp.createContact(row[0], '',
row[1]);
 newContact.addPhone(ContactsApp.Field.MOBILE_PHONE,
row[2]);
```

```
  }
  });
}
```

Each of these exercises offers a practical scenario to apply Google Apps Script for solving real-world problems, enhancing your scripting skills, and automating tasks within the Google Workspace ecosystem.