🌟 Begin Your Journey with Google Apps Script! 🌟

50 *Coding* Exercises

Dive Deeper into Google Apps Script with New Challenges!

💡 Embark on your scripting journey today and discover the endless possibilities!

🔥 Hot off the press: 50 coding exercises for Google Apps Script enthusiasts! These exercises delve into advanced scripting techniques, from automating Google Workspace to integrating with Google Cloud services. 🛠️ Whether you're looking to automate complex tasks, build interactive web apps, or integrate with APIs, these exercises provide the perfect platform to level up your scripting skills.

🚀 Take on these challenges and showcase your evolving Google Apps Script expertise! These exercises cover everything from creating your first script to automating tasks across Google Workspace.

👩‍💻 Perfect for those new to coding or looking to expand their Google Workspace skills, these exercises are your first step towards mastering Google Apps Script.

👩‍💻 Whether you're looking to streamline your workflows or build powerful integrations across Google Workspace, these exercises offer practical experience to boost your skills.

💡 Embark on these new challenges and enhance your Google Apps Script knowledge!

#GoogleAppsScript #BeginnerCoding #LearnToCode #Scripting #GoogleWorkspace #AutomateTasks #CodingExercises #TechSkills #SpreadsheetAutomation #DocumentAutomation #GoogleSheets #GoogleDocs #GoogleCalendar #DeveloperTools #AdvancedCoding #ScriptingChallenges #Automation #GoogleCloud #APIIntegration #WebApps #DeveloperJourney #DriveAPI #SheetsAPI #GmailAPI #CalendarAPI #TechLearning #CodingJourney

Exercise 1: Hello World in Google Apps Script

Objective: Write a simple script to log "Hello, World!".

Explanation: This is the most basic exercise to get started with Google Apps Script. It introduces the Logger class.

Code:

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

Exercise 2: Create a Custom Function for Google Sheets

Objective: Create a custom function that doubles a number.

Explanation: Custom functions can be used like regular functions in Google Sheets. This exercise introduces creating a simple custom function.

Code:

```
function doubleNumber(number) {
  return number * 2;
}
// Usage in Sheets: =doubleNumber(10)
```

Exercise 3: Update Spreadsheet Data

Objective: Write a script to modify a cell in Google Sheets.

Explanation: This exercise introduces SpreadsheetApp and how to programmatically modify spreadsheet data.

Code:

```
function updateSpreadsheet() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  sheet.getRange('A1').setValue('Updated!');
}
```

Exercise 4: Read Data from a Spreadsheet

Objective: Read and log data from a specific cell.

Explanation: Teaches how to read data from a spreadsheet, an essential skill for data manipulation.

Code:

```
function readSpreadsheetData() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var value = sheet.getRange('A1').getValue();
  Logger.log(value);
}
```

Exercise 5: Create a New Spreadsheet

Objective: Use Google Apps Script to create a new spreadsheet.

Explanation: Introduces the concept of creating new files (spreadsheets) using Google Apps Script.

Code:

```
function createNewSpreadsheet() {
  var newSpreadsheet = SpreadsheetApp.create("New Spreadsheet");
  Logger.log(newSpreadsheet.getUrl());
}
```

Exercise 6: Send an Email

Objective: Send a simple email using GmailApp.

Explanation: Demonstrates the use of GmailApp to send emails, a practical application of Google Apps Script.

Code:

```
function sendEmail() {
  GmailApp.sendEmail('recipient@example.com', 'Test Email', 'Hello from Google Apps Script!');
}
```

Exercise 7: Fetch Calendar Events

Objective: Fetch and log upcoming events from the user's calendar.

Explanation: Introduces CalendarApp to interact with Google Calendar.

Code:
```
function fetchCalendarEvents() {
  var events = CalendarApp.getDefaultCalendar().getEvents(new Date(),
new Date(new Date().getTime() + (7 * 24 * 60 * 60 * 1000)));
  for (var i = 0; i < events.length; i++) {
    Logger.log(events[i].getTitle());
  }
}
```

Exercise 8: Create a Google Document

Objective: Create a new Google Document using Google Apps Script.

Explanation: Shows how to programmatically create a new document.

Code:
```
function createGoogleDoc() {
  var doc = DocumentApp.create('New Document');
  Logger.log(doc.getUrl());
}
```

Exercise 9: Add Content to a Google Document

Objective: Add text content to a newly created Google Document.

Explanation: Builds on the previous exercise, showing how to add content to a document.

Code:

```
function addContentToDoc() {
  var doc = DocumentApp.create('New Document');
  var body = doc.getBody();
  body.appendParagraph('This is some sample text!');
}
```

Exercise 10: Format a Google Spreadsheet Cell

Objective: Apply formatting (like font size, background color) to a cell in Google Sheets.

Explanation: Introduces basic cell formatting in Google Sheets through scripting.

Code:

```
function formatCell() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var cell = sheet.getRange('A1');
  cell.setFontSize(12);
  cell.setBackground('yellow');
}
```

Exercise 11: Add Custom Menu in Google Sheets

Objective: Create a custom menu in Google Sheets with a script function.

Explanation: Learn how to enhance the Google Sheets UI by adding a custom menu that triggers script functions.

Code:

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom Menu')
```

```
    .addItem('Run Function', 'sampleFunction')
    .addToUi();
}

function sampleFunction() {
  SpreadsheetApp.getUi().alert('Function Executed!');
}
```

Exercise 12: Import Data from Another Spreadsheet

Objective: Write a script to import data from another spreadsheet.

Explanation: Demonstrates how to read and import data from one spreadsheet to another.

Code:

```
function importData() {
  var sourceSpreadsheetID = 'source-spreadsheet-id'; // Replace with source spreadsheet ID
  var sourceSheetName = 'Sheet1'; // Replace with source sheet name
  var targetSheetName = 'Sheet1'; // Replace with target sheet name

  var sourceSpreadsheet = SpreadsheetApp.openById(sourceSpreadsheetID);
  var sourceSheet = sourceSpreadsheet.getSheetByName(sourceSheetName);
  var data = sourceSheet.getDataRange().getValues();

  var targetSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(targetSheetName);
  targetSheet.getRange(1, 1, data.length, data[0].length).setValues(data);
}
```

Exercise 13: Generate a Report and Email as PDF

Objective: Create a report in Google Sheets and email it as a PDF.

Explanation: Learn how to programmatically generate a PDF from a Google Sheet and send it via email.

Code:

```
function emailSpreadsheetAsPDF() {
  var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = spreadsheet.getSheetByName('Sheet1'); // Update sheet
name
  var url = 'https://docs.google.com/spreadsheets/d/' + spreadsheet.getId()
+ '/export?exportFormat=pdf&gid=' + sheet.getSheetId();
  var response = UrlFetchApp.fetch(url, {
    headers: { 'Authorization': 'Bearer ' + ScriptApp.getOAuthToken() }
  });

  var recipient = 'recipient@example.com'; // Replace with actual email
address
  var subject = 'Spreadsheet Report';
  var body = 'Find the attached report.';
  GmailApp.sendEmail(recipient, subject, body, {
    attachments: [response.getBlob()]
  });
}
```

Exercise 14: Create a Directory in Google Drive

Objective: Write a script to create a new directory (folder) in Google Drive.

Explanation: Demonstrates how to use the DriveApp service to create new folders in Google Drive.

Code:

```
function createDirectory() {
  var folderName = 'New Folder';
  DriveApp.createFolder(folderName);
}
```

Exercise 15: Retrieve and Log Drive File Information

Objective: Fetch and log details of a file in Google Drive.

Explanation: Teaches how to retrieve metadata for a file stored in Google Drive.

Code:

```
function logDriveFileInfo() {
  var fileId = 'file-id'; // Replace with actual file ID
  var file = DriveApp.getFileById(fileId);
  Logger.log('Name: ' + file.getName());
  Logger.log('Size: ' + file.getSize());
}
```

Exercise 16: Create Google Calendar Event

Objective: Use a script to create an event in Google Calendar.

Explanation: Shows how to interact with Google Calendar to create events.

Code:

```
function createCalendarEvent() {
  var calendar = CalendarApp.getDefaultCalendar();
  var title = 'Meeting';
  var startTime = new Date('2024-03-01T09:00:00');
  var endTime = new Date('2024-03-01T10:00:00');
  calendar.createEvent(title, startTime, endTime);
}
```

Exercise 17: Translate Text

Objective: Translate text from one language to another using LanguageApp.

Explanation: Introduces the Language Service for translating text.

Code:

```
function translateText() {
  var text = 'Hello, world!';
  var translatedText = LanguageApp.translate(text, 'en', 'es'); // Translate from English to Spanish
  Logger.log(translatedText);
}
```

Exercise 18: Parse JSON Data

Objective: Fetch and parse JSON data from a URL.

Explanation: Demonstrates how to use URLFetchApp to call an API and parse JSON data.

Code:

```
function fetchAndParseJSON() {
  var url = 'https://api.example.com/data'; // Replace with actual API URL
  var response = UrlFetchApp.fetch(url);
  var jsonData = JSON.parse(response.getContentText());
  Logger.log(jsonData);
}
```

Exercise 19: Automatically Backup Spreadsheet Daily

Objective: Create a time-driven trigger to backup a spreadsheet every day.

Explanation: Shows how to set up automatic triggers for daily tasks.

Code:
```
function createBackupTrigger() {
  ScriptApp.newTrigger('backupSpreadsheet')
          .timeBased()
          .everyDays(1)
          .atHour(1) // Runs at 1 AM every day
          .create();
}

function backupSpreadsheet() {
  var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
  var backupFolderId = 'folder-id'; // Replace with your backup folder ID
  DriveApp.getFileById(spreadsheet.getId()).makeCopy('Backup - ' + new
Date(), DriveApp.getFolderById(backupFolderId));
}
```

Exercise 20: Log Email Subjects from Gmail

Objective: Retrieve and log the subject lines of the latest emails in Gmail.

Explanation: Introduces GmailApp to interact with Gmail, specifically fetching email subjects.

Code:
```
function logEmailSubjects() {
  var threads = GmailApp.getInboxThreads(0, 10); // Fetch first 10 threads
  for (var i = 0; i < threads.length; i++) {
    var subject = threads[i].getFirstMessageSubject();
    Logger.log(subject);
  }
}
```

Exercise 21: Dynamically Generate a Drop-down List in Google Sheets

Objective: Use Google Apps Script to create a drop-down list in a cell based on data in another range.

Explanation: Learn how to use data validation in Google Sheets with scripting to create dynamic drop-down lists.

Code:

```
function createDropdown() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange("A1:A10"); // Range to turn into drop-downs
  var dropdownRange = sheet.getRange("B1:B10"); // Range to use as drop-down source

  var rule = SpreadsheetApp.newDataValidation().requireValueInRange(dropdownRange).build();
  range.setDataValidation(rule);
}
```

Exercise 22: Extract Email Addresses from Gmail to Google Sheets

Objective: Write a script that extracts email addresses from Gmail messages and stores them in a Google Sheet.

Explanation: Develop the ability to interact with Gmail messages and parse data into a spreadsheet.

Code:

```
function extractEmailsToSheet() {
```

```
  var threads = GmailApp.getInboxThreads(0, 5); // Get the first 5 email
threads
  var sheet = SpreadsheetApp.create("Emails Extracted").getActiveSheet();
  var emails = [];

  threads.forEach(function(thread) {
    var messages = thread.getMessages();
    messages.forEach(function(message) {
      emails.push(message.getFrom());
    });
  });

  sheet.getRange(1, 1, emails.length, 1).setValues(emails.map(email =>
[email]));
}
```

Exercise 23: Create a Folder Tree in Google Drive

Objective: Use DriveApp to create a nested folder structure in Google Drive.

Explanation: Learn to programmatically create and organize folders in Google Drive.

Code:

```
function createFolderTree() {
  var parentFolder = DriveApp.createFolder('Parent Folder');
  var childFolder1 = parentFolder.createFolder('Child Folder 1');
  var childFolder2 = parentFolder.createFolder('Child Folder 2');
  childFolder1.createFolder('Sub Child Folder 1');
  childFolder2.createFolder('Sub Child Folder 2');
}
```

Exercise 24: Post a Message to Google Chat

Objective: Send a simple text message to a Google Chat room using Google Apps Script.

Explanation: Introduce interaction with Google Chat for notifications or updates.

Code:
```
function postMessageToGoogleChat() {
  var webhookUrl = 'YOUR-WEBHOOK-URL'; // Replace with your webhook URL
  var message = {
    'text': 'Hello from Google Apps Script!'
  };

  UrlFetchApp.fetch(webhookUrl, {
    'method': 'post',
    'contentType': 'application/json',
    'payload': JSON.stringify(message)
  });
}
```

Exercise 25: Analyze Text Sentiment with Google Natural Language API

Objective: Use the Google Natural Language API to analyze the sentiment of a text.

Explanation: Expand the capabilities by integrating with Google Cloud services like the Natural Language API.

Code:
```
function analyzeSentiment() {
  var apiKey = 'YOUR-API-KEY'; // Replace with your API Key
```

```
  var text = 'I love Google Apps Script!';
  var apiEndpoint =
'https://language.googleapis.com/v1/documents:analyzeSentiment?key=' +
apiKey;
  var document = {
    'document': {
      'type': 'PLAIN_TEXT',
      'content': text
    }
  };

  var response = UrlFetchApp.fetch(apiEndpoint, {
    'method': 'post',
    'contentType': 'application/json',
    'payload': JSON.stringify(document)
  });

  var sentiment =
JSON.parse(response.getContentText()).documentSentiment;
  Logger.log('Score: ' + sentiment.score + ', Magnitude: ' +
sentiment.magnitude);
}
```

Exercise 26: Track Google Form Responses in Real-time

Objective: Write a script that logs Google Form responses as they are submitted.

Explanation: Learn to use triggers to respond to real-time events like form submissions.

Code:

```
function setUpTrigger() {
```

```
  var form = FormApp.openById('FORM-ID'); // Replace with your form ID
  ScriptApp.newTrigger('logResponse')
          .forForm(form)
          .onFormSubmit()
          .create();
}

function logResponse(e) {
  var responses = e.values;
  Logger.log(responses);
}
```

Exercise 27: Create a Simple Web App Interface

Objective: Use HTML Service to create a basic web application interface.

Explanation: Develop skills in creating web-based interfaces with Google Apps Script.

Code:

```
function doGet() {
  return HtmlService.createHtmlOutput('<h1>Welcome to my Web App!</h1>');
}
```

Exercise 28: Batch Update Google Calendar Events

Objective: Modify multiple Google Calendar events in one operation.

Explanation: Learn to efficiently update several calendar events, reducing the number of API calls.

Code:

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

```
  var events = calendar.getEvents(new Date('2024-01-01T00:00:00Z'), new
Date('2024-01-31T23:59:59Z'));

  events.forEach(function(event) {
    event.setTitle('Updated Event Title');
  });
}
```

Exercise 29: Auto-generate Document based on Form Responses

Objective: Create a Google Document for each new Google Form response.

Explanation: Combine Google Forms, Docs, and Script to automate document creation.

Code:

```
function onFormSubmit(e) {
  var responses = e.namedValues;
  var doc = DocumentApp.create('Response for ' + responses['Name'][0]);
  var body = doc.getBody();
  body.appendParagraph('Name: ' + responses['Name'][0]);
  body.appendParagraph('Email: ' + responses['Email'][0]);
}
```

Exercise 30: Schedule Email Reminders for Spreadsheet Tasks

Objective: Send scheduled email reminders based on due dates in a Google Spreadsheet.

Explanation: Implement a time-driven script to automate email reminders for tasks or events.

Code:

```
function scheduleEmailReminders() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Tasks'); //
Assume 'Tasks' sheet exists
  var tasks = sheet.getRange('A2:B' + sheet.getLastRow()).getValues(); //
Assume tasks are in columns A and B

  tasks.forEach(function(task) {
    var taskName = task[0];
    var dueDate = new Date(task[1]);
    if (isDueTomorrow(dueDate)) {
      GmailApp.sendEmail('recipient@example.com', 'Reminder for ' +
taskName, 'Task ' + taskName + ' is due tomorrow!');
    }
  });
}

function isDueTomorrow(date) {
  var today = new Date();
  var tomorrow = new Date();
  tomorrow.setDate(today.getDate() + 1);
  return date.getDate() === tomorrow.getDate() &&
         date.getMonth() === tomorrow.getMonth() &&
         date.getFullYear() === tomorrow.getFullYear();
}
```

Exercise 31: Generate a Custom Report in Google Sheets

Objective: Create a script that generates a report based on data in a spreadsheet.

Explanation: Learners will understand how to manipulate spreadsheet data to create a summarized report.

Code:
```
function generateReport() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var dataRange = sheet.getDataRange();
  var data = dataRange.getValues();

  // Process data and create a report
  // For example, summarize data, count items, etc.
}
```

Exercise 32: Create a Directory and Subdirectories in Google Drive

Objective: Write a script to create a main directory and several subdirectories within it.

Explanation: This exercise aims to teach directory structure management within Google Drive using Google Apps Script.

Code:
```
function createDirectoryWithSubdirectories() {
  var mainFolder = DriveApp.createFolder('Main Folder');
  var subfolders = ['Subfolder 1', 'Subfolder 2', 'Subfolder 3'];

  subfolders.forEach(function(name) {
```

```
    mainFolder.createFolder(name);
  });
}
```

Exercise 33: Insert a Chart in Google Sheets

Objective: Use Google Apps Script to create and insert a chart in a spreadsheet.

Explanation: Teaches how to programmatically create charts in Google Sheets based on data ranges.

Code:

```
function insertChart() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange('A1:B10'); // Assuming data exists in this range
  var chart = sheet.newChart()
                   .setChartType(Charts.ChartType.LINE)
                   .setRange(range)
                   .setPosition(5, 5, 0, 0)
                   .build();
  sheet.insertChart(chart);
}
```

Exercise 34: Append Row to Another Spreadsheet

Objective: Write a script to append a row of data to another spreadsheet.

Explanation: Demonstrates how to work with multiple spreadsheets and transfer data between them.

Code:

```
function appendRowToAnotherSpreadsheet() {
  var sourceSpreadsheet = SpreadsheetApp.getActiveSpreadsheet();
```

```
  var sourceSheet = sourceSpreadsheet.getSheetByName("Source");
  var rowData = sourceSheet.getRange("A1:D1").getValues(); // Example
row data

  var targetSpreadsheetId = 'target-spreadsheet-id'; // Replace with your
target spreadsheet ID
  var targetSpreadsheet =
SpreadsheetApp.openById(targetSpreadsheetId);
  var targetSheet = targetSpreadsheet.getSheetByName("Target");

  targetSheet.appendRow(rowData[0]);
}
```

Exercise 35: Delete Rows Based on Condition

Objective: Remove rows from a spreadsheet based on a specific condition.

Explanation: Helps understand how to iterate over rows and delete them based on certain criteria.

Code:

```
function deleteRowsBasedOnCondition() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var rows = sheet.getDataRange().getValues();

  rows.forEach(function(row, index) {
    if (row[0] === 'Delete') { // Assuming the condition is in the first column
      sheet.deleteRow(index + 1);
      // Note: When deleting rows, consider iterating backwards to avoid
index shifting issues
    }
  });
}
```

Exercise 36: Retrieve File Info from a Google Drive Folder

Objective: List all files in a specific Google Drive folder and log their details.

Explanation: Teaches how to access and retrieve information about files stored in Google Drive.

Code:

```
function listFilesInFolder() {
  var folderId = 'your-folder-id'; // Replace with your folder ID
  var folder = DriveApp.getFolderById(folderId);
  var files = folder.getFiles();

  while (files.hasNext()) {
    var file = files.next();
    Logger.log('File Name: ' + file.getName() + ', File ID: ' + file.getId());
  }
}
```

Exercise 37: Modify Google Calendar Event Details

Objective: Update the details (like title, time) of a specific Google Calendar event.

Explanation: Demonstrates how to find and update events in Google Calendar.

Code:

```
function updateCalendarEvent() {
  var calendar = CalendarApp.getDefaultCalendar();
  var events = calendar.getEventsForDay(new Date()); // Get today's events
  if (events.length > 0) {
    var event = events[0]; // Update the first event
    event.setTitle('Updated Event Title');
```

```
    event.setTime(new Date('2024-01-01T10:00:00Z'), new
Date('2024-01-01T11:00:00Z'));
  }
}
```

Exercise 38: Create a Simple Approval Workflow

Objective: Build a script for a basic approval workflow using Google Sheets and Gmail.

Explanation: Combines Google Sheets and Gmail to create an interactive approval process.

Code:

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

  data.forEach(function(row, index) {
    if (row[3] === 'Pending') { // Assuming approval status is in column 4
      var email = row[2]; // Assuming email is in column 3
      GmailApp.sendEmail(email, 'Approval Request', 'Please review your
request.');
      sheet.getRange(index + 1, 4).setValue('Requested'); // Update status
    }
  });
}
```

Exercise 39: Extract and Summarize Data from Emails

Objective: Parse data from Gmail messages and summarize it in Google Sheets.

Explanation: Shows how to read Gmail messages, extract information, and use it to populate a spreadsheet.

Code:

```
function summarizeEmailData() {
  var threads = GmailApp.search('label:your-label'); // Replace with your search criteria
  var sheet = SpreadsheetApp.create("Email Summary").getActiveSheet();
  var summaries = [];

  threads.forEach(function(thread) {
    var messages = thread.getMessages();
    messages.forEach(function(message) {
      var content = message.getPlainBody();
      // Process content to extract data
      // For example, extract specific information from the email body
      summaries.push([content]); // Add extracted data to summaries array
    });
  });

  sheet.getRange(1, 1, summaries.length, 1).setValues(summaries);
}
```

Exercise 40: Sync Contacts from Google Contacts to a Spreadsheet

Objective: Retrieve contacts from Google Contacts and list them in a Google Sheet.

Explanation: Teaches how to integrate with Google Contacts and manage data in a spreadsheet.

Code:

```
function syncGoogleContacts() {
```

```
  var contacts = ContactsApp.getContacts();
  var sheet = SpreadsheetApp.create("Contacts Sync").getActiveSheet();
  var data = [];

  contacts.forEach(function(contact) {
    var email = (contact.getEmails()[0]) ? contact.getEmails()[0].getAddress()
: '';
    var phone = (contact.getPhones()[0]) ?
contact.getPhones()[0].getPhoneNumber() : '';
    data.push([contact.getFullName(), email, phone]);
  });

  sheet.getRange(1, 1, data.length, 3).setValues(data);
}
```

Exercise 41: Automate Document Creation Based on Spreadsheet Data

Objective: Generate a Google Document for each row in a Google Sheet.

Explanation: Shows how to automate document creation using data stored in a spreadsheet.

Code:

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

  data.forEach(function(row, index) {
    if (index === 0) return; // Skip header row
    var doc = DocumentApp.create('Document for ' + row[0]); // Assuming
first column has a unique identifier
    var body = doc.getBody();
```

```
    body.appendParagraph('Data for this document: ' + row.join(', '));
    // Additional formatting and content addition here
  });
}
```

Exercise 42: Sync Spreadsheet Data with Calendar

Objective: Create calendar events based on data in a Google Sheet.

Explanation: Integrates Google Sheets and Calendar to automate event creation.

Code:

```
function createEventsFromSheet() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var data = sheet.getDataRange().getValues();
  var calendar = CalendarApp.getDefaultCalendar();

  data.forEach(function(row, index) {
    if (index === 0) return; // Skip header row
    var title = row[0];
    var startTime = new Date(row[1]);
    var endTime = new Date(row[2]);
    calendar.createEvent(title, startTime, endTime);
    // Assume columns have event title, start time, and end time
  });
}
```

Exercise 43: Batch Resize Images in Google Drive

Objective: Resize all images in a specific Google Drive folder.

Explanation: Demonstrates how to manipulate and modify image files stored in Google Drive.

Code:

```
function resizeDriveImages() {
  var folder = DriveApp.getFolderById('folder-id'); // Replace with the folder ID
  var images = folder.getFilesByType(MimeType.JPEG); // Adjust the MIME type as needed

  while (images.hasNext()) {
    var image = images.next();
    var blob = image.getBlob();
    var resizedImage = ImagesService.newImage(blob).resize(200, 200).getBlob(); // Resize to 200x200
    folder.createFile(resizedImage).setName('Resized-' + image.getName());
  }
}
```

Exercise 44: Create a Custom Google Form with Script

Objective: Programmatically create a Google Form with multiple question types.

Explanation: Teaches how to use Google Apps Script to automate the creation of Google Forms.

Code:

```
function createCustomForm() {
  var form = FormApp.create('New Form');
  form.addTextItem().setTitle('Name');
  form.addMultipleChoiceItem().setTitle('Favorite Color')
      .setChoiceValues(['Red', 'Blue', 'Green']);
  form.addParagraphTextItem().setTitle('Feedback');
  // More form configurations here
}
```

Exercise 45: Monitor Drive Folder and Send Email Alerts for New Files

Objective: Send an email alert whenever a new file is added to a specific Google Drive folder.

Explanation: Shows how to create a script that monitors changes in a Drive folder and sends notifications.

Code:

```
function monitorDriveFolder() {
  var folderId = 'folder-id'; // Replace with your folder ID
  var folder = DriveApp.getFolderById(folderId);
  var files = folder.getFiles();

  while (files.hasNext()) {
    var file = files.next();
    // Check if the file was added recently or meets certain criteria
    // If yes, send an email notification
    GmailApp.sendEmail('recipient@example.com', 'New file added', 'A new file ' + file.getName() + ' was added.');
  }
}
```

Exercise 46: Automate Responses to Google Form Submissions

Objective: Send a custom email response when a Google Form is submitted.

Explanation: Integrates Google Forms and Gmail for automated email responses based on form submissions.

Code:

```
function setUpFormTrigger() {
  var form = FormApp.openById('form-id'); // Replace with your form ID
  ScriptApp.newTrigger('onFormSubmit')
          .forForm(form)
          .onFormSubmit()
          .create();
}

function onFormSubmit(e) {
  var responses = e.values;
  var email = responses[1]; // Assuming email is the second question
  GmailApp.sendEmail(email, 'Thank you for your submission', 'We
received your response.');
}
```

Exercise 47: Generate a PDF Invoice from Spreadsheet Data

Objective: Create a PDF invoice using data from a Google Sheet.

Explanation: Teaches how to format and export spreadsheet data as a PDF.

Code:

```
function generatePdfInvoice() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getDataRange();
  var values = range.getValues();

  // Format the sheet as needed for the invoice
  // Then export as PDF
  var pdf =
DriveApp.getFileById(sheet.getParent().getId()).getAs('application/pdf');
  DriveApp.createFile(pdf).setName('Invoice.pdf');
```

```
}
```

Exercise 48: Parse and Analyze Email Contents

Objective: Read emails from Gmail and extract specific information.

Explanation: Focuses on parsing email content for data analysis or extraction.

Code:

```
function analyzeEmails() {
  var threads = GmailApp.getInboxThreads();
  threads.forEach(function(thread) {
    var messages = thread.getMessages();
    messages.forEach(function(message) {
      var content = message.getPlainBody();
      // Process and analyze content
      // For example, extract dates, names, or other specific information
    });
  });
}
```

Exercise 49: Auto-Generate Calendar Events from Spreadsheet Schedules

Objective: Create Google Calendar events based on a schedule stored in a Google Sheet.

Explanation: Demonstrates synchronization between Google Sheets and Google Calendar.

Code:

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

```
  var calendar = CalendarApp.getDefaultCalendar();

  data.forEach(function(row, index) {
    if (index === 0) return; // Skip header row
    var eventTitle = row[0];
    var startTime = new Date(row[1]);
    var endTime = new Date(row[2]);
    calendar.createEvent(eventTitle, startTime, endTime);
  });
}
```

Exercise 50: Sync Contact Information from Google Contacts to Sheets

Objective: Retrieve contacts from Google Contacts and export them to a Google Sheet.

Explanation: Teaches how to integrate Google Contacts with Google Sheets for data synchronization.

Code:

```
function exportContactsToSheet() {
  var contacts = ContactsApp.getContacts();
  var sheet = SpreadsheetApp.create("Contacts Export").getActiveSheet();
  var data = contacts.map(function(contact) {
    return [contact.getFullName(), contact.getEmails().map(email =>
email.getAddress()).join(", ")];
  });

  sheet.getRange(1, 1, data.length, 2).setValues(data);
}
```

These exercises cover a wide range of functionalities and applications, providing practical experience in scripting for Google Apps Script and integrating with various Google services.