Google Apps Script Code Exercises Advanced

Exercise 11: Email Auto-Responder

Code:

```
function autoResponder() {

  var threads = GmailApp.getInboxThreads(0, 1); // Get
  the latest email thread



  if (threads.length > 0) {

    var senderEmail =
  threads[0].getMessages()[0].getFrom();

    var customMessage = getCustomMessage(senderEmail);



    // Send auto-reply

    GmailApp.sendEmail(senderEmail, 'Auto-Reply',
  customMessage);

  }

}
```

```
function getCustomMessage(senderEmail) {

  // Implement logic to fetch a custom message based on
  sender's email

  // For simplicity, a static message is returned here

  return 'Thank you for your email. We will get back to
  you shortly!';

}
```

Details:

1. The autoResponder function checks the latest email thread and sends an auto-reply to the sender.
2. The getCustomMessage function retrieves a custom message based on the sender's email (you can customize this logic).

Exercise 12: Spreadsheet Data Analysis

Code:

```
function analyzeSpreadsheetData() {
  var spreadsheet =
  SpreadsheetApp.openById('your_spreadsheet_id');

  var sheet = spreadsheet.getSheetByName('Sheet1');



  var data = sheet.getDataRange().getValues();

  var statistics = calculateStatistics(data);



  Logger.log('Statistics:', statistics);

}



function calculateStatistics(data) {

  // Implement logic to calculate statistics (e.g.,
  average, sum, etc.)

  // For simplicity, calculating average of numerical
  values in the first column
  var numericalValues = data.map(row =>
  row[0]).filter(value => !isNaN(value));

  var average = numericalValues.reduce((sum, value) =>
  sum + value, 0) / numericalValues.length;



  return { average: average };

}
```

Details:

1. The analyzeSpreadsheetData function retrieves data from a specified spreadsheet and calculates statistics.
2. The calculateStatistics function calculates the average of numerical values in the first column (you can customize this logic).

Exercise 13: Calendar Event Synchronization

Code:

```
function syncEventsWithCalendar() {
```

```
var spreadsheet =
SpreadsheetApp.openById('your_spreadsheet_id');

var sheet = spreadsheet.getSheetByName('Events');



var events = sheet.getDataRange().getValues();



for (var i = 1; i < events.length; i++) {

  try {

    var event =
CalendarApp.getDefaultCalendar().createEvent(events[i][
0], new Date(events[i][1]), new Date(events[i][2]));

    Logger.log('Event created:', event.getTitle());

  } catch (error) {

    Logger.log('Error creating event:', error);

  }
  }

}
```

Details:

1. The syncEventsWithCalendar function syncs events from a specified spreadsheet with the default Google Calendar.
2. It logs successful event creations and errors with details.

Exercise 14: Gmail Label Automation

Code:

```
function labelAutomation() {

  var threads = GmailApp.getInboxThreads();



  for (var i = 0; i < threads.length; i++) {

    var labels = getLabelsForThread(threads[i]);

    threads[i].addLabels(labels);
  }

}



function getLabelsForThread(thread) {

  // Implement logic to determine labels based on
  predefined rules

  // For simplicity, returning a static set of labels
  here

  return ['Important', 'Customer Inquiry'];

}
```

Details:

1. The labelAutomation function labels incoming emails based on predefined rules.
2. The getLabelsForThread function determines labels for a specific thread (you can customize this logic).

Exercise 15: Data Validation in Google Forms

Code:

```
function formResponseValidation(e) {

  var responses = e.values;



  // Implement data validation rules

  var isValid = validateResponses(responses);



  if (!isValid) {

    // Send an alert for invalid entries

    sendAlertEmail('Invalid Form Entry', 'Please review
  the form responses for validation issues.');

  }

}
```

```
function validateResponses(responses) {

  // Implement validation rules based on form responses

  // For simplicity, checking if the first field is not
  empty

  return responses[0] !== '';

}



function sendAlertEmail(subject, body) {

  // Implement logic to send an alert email

  // For simplicity, using GmailApp to send the email

  GmailApp.sendEmail('admin@example.com', subject, body);

}
```

Details:

1. The formResponseValidation function validates responses to a Google Form.
2. The validateResponses function implements validation rules (you can customize this logic).
3. An alert email is sent for invalid entries using the sendAlertEmail function.

Exercise 16: Document Merge from Google Form Responses

Code:

```
function mergeDocumentsFromFormResponses() {

  var form = FormApp.openById('your_form_id');

  var responses = form.getResponses();



  for (var i = 0; i < responses.length; i++) {

    var response = responses[i];
    var responseData =
  response.getItemResponses().map(item =>
  item.getResponse());

    createMergedDocument(responseData);

  }

}



function createMergedDocument(data) {

  // Implement logic to create a merged document using
  data

  // For simplicity, creating a new Google Doc with data

  var doc = DocumentApp.create('Merged Document');

  var body = doc.getBody();



  // Insert data into the document
  body.appendParagraph('Name: ' + data[0]);

  body.appendParagraph('Age: ' + data[1]);

  // ... add more data



  Logger.log('Merged document created:', doc.getUrl());

}
```

Details:

1. The mergeDocumentsFromFormResponses function fetches responses from a Google Form and creates merged documents.
2. The createMergedDocument function creates a new Google Doc with data from form responses.

Exercise 17: Interactive Google Sheets Dashboard

Code:

```
function updateDashboard() {

  var spreadsheet =
  SpreadsheetApp.openById('your_dashboard_spreadsheet_id'
  );

  var dataSheet = spreadsheet.getSheetByName('Data');

  var dashboardSheet =
  spreadsheet.getSheetByName('Dashboard');



  var data = dataSheet.getDataRange().getValues();



  // Implement logic to update the dashboard based on
  real-time data

  // For simplicity, updating a chart with data from the
  first two columns

  var chartBuilder =
  dashboardSheet.newChart().asColumnChart();
  chartBuilder.addRange(dataSheet.getRange('A:B'));



  var chart = chartBuilder.build();

  dashboardSheet.updateChart(chart);

}
```

Details:

1. The updateDashboard function updates an interactive dashboard in a Google Sheets.
2. The logic for updating the dashboard is simplified here (you can customize this logic).

Exercise 18: Gmail Attachment Manager

Code:

```
function manageAttachments() {

  var threads = GmailApp.getInboxThreads();
  for (var i = 0; i < threads.length; i++) {

    var attachments =
  getAttachmentsFromThread(threads[i]);


    for (var j = 0; j < attachments.length; j++) {

      saveAttachmentToDrive(attachments[j]);

    }

  }

}


function getAttachmentsFromThread(thread) {

  // Implement logic to fetch attachments from a thread

  // For simplicity, returning static attachments here

  return thread.getMessages()[0].getAttachments();
```

```
}

function saveAttachmentToDrive(attachment) {

  // Implement logic to save attachment to Google Drive

  // For simplicity, logging the file name

  Logger.log('Attachment saved to Drive:',
  attachment.getName());

}
```

Details:

1. The manageAttachments function fetches attachments from emails and saves them to Google Drive.
2. The getAttachmentsFromThread function retrieves attachments from a specific email thread.
3. The saveAttachmentToDrive function saves the attachment to Google Drive (you can customize this logic).

Exercise 19: User Authentication and Authorization

Code:

```
function userAuthenticationAndAuthorization() {

  var user = authenticateUser();



  if (user) {

    // User is authenticated, proceed with authorization
  based on roles

    authorizeUser(user);

  } else {

    // User authentication failed, handle accordingly

    Logger.log('Authentication failed');

  }
```

```
}



function authenticateUser() {

  // Implement logic for user authentication

  // For simplicity, assuming authentication is
  successful

  return { username: 'john_doe', roles: ['admin'] };

}



function authorizeUser(user) {

  // Implement logic for user authorization based on
  roles

  // For simplicity, checking if the user has admin role

  if (user.roles.includes('admin')) {
    Logger.log('User authorized as admin');

  } else {

    Logger.log('User not authorized');

  }

}
```

Details:

1. The userAuthenticationAndAuthorization function handles user authentication and authorization.
2. The authenticateUser function simulates user authentication (you can customize this logic).

The authorizeUser function checks user roles and authorizes accordingly.

Exercise 20: External API Integration

Code:

function fetchDataFromExternalAPI() {

```
  var apiUrl = 'https://api.example.com/data';


  try {

    var response = UrlFetchApp.fetch(apiUrl);

    var data = JSON.parse(response.getContentText());


    // Implement logic to update a Google Spreadsheet with the
    retrieved data

    updateSpreadsheetWithData(data);

  } catch (error) {

    Logger.log('Error fetching data from API:', error);

  }

}
```

```
function updateSpreadsheetWithData(data) {

  var spreadsheet =
    SpreadsheetApp.openById('your_spreadsheet_id');

  var sheet = spreadsheet.getSheetByName('Data');



  // Implement logic to update the spreadsheet with data

  // For simplicity, updating the first row with data

  sheet.getRange(1, 1, 1, data.length).setValues([data]);

}
```

Details:

1. The fetchDataFromExternalAPI function fetches data from an external API.
2. The updateSpreadsheetWithData function updates a Google Spreadsheet with the retrieved data.