🌐 *Supercharge Your Google Workspace*: Engaging **Google Apps Script** Projects! 💻🚀

Exercise 1: Create a Custom Menu in Google Sheets

Objective: To create a custom menu in Google Sheets that triggers a simple script.

Code Sample:

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom Menu')
      .addItem('Show Alert', 'showAlert')
      .addToUi();
}

function showAlert() {
  SpreadsheetApp.getUi().alert('Hello, Google
Sheets!');
}
```

Explanation:

This script creates a custom menu named "Custom Menu" in the Google Sheets UI. When "Show Alert" is clicked, it triggers the showAlert function that displays a simple alert box with a message.

Exercise 2: Read Data from a Range in Google Sheets

Objective: Read data from a specified range in a sheet and log it.

Code Sample:

```
function readRangeData() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange("A1:B2");
  var values = range.getValues();

  for (var i = 0; i < values.length; i++) {
    var row = values[i];
    Logger.log(row[0] + ", " + row[1]);
  }
}
```

Explanation:

This script reads data from the range A1:B2 of the active sheet. getValues returns a two-dimensional array of values, which is then iterated over to log each cell's data.

Exercise 3: Send an Email using GmailApp

Objective: Use Google Apps Script to send an email via Gmail.

Code Sample:

```
function sendEmail() {
  var recipient = "example@example.com";
  var subject = "Test Email from Google Apps Script";
  var body = "This is a test email sent from Google
Apps Script.";

  GmailApp.sendEmail(recipient, subject, body);
}
```

Explanation:

This script sends an email using the Gmail service. Replace recipient with the desired email address. The GmailApp.sendEmail method is used to send the email.

Exercise 4: Create a Google Doc

Objective: Programmatically create a Google Doc with some content.

Code Sample:

```
function createDocument() {
  var doc = DocumentApp.create('New Document');
  var body = doc.getBody();
```

```
  body.appendParagraph('This is a new Google Document
created by Google Apps Script.');

  doc.saveAndClose();
}
```

Explanation:

This script creates a new Google Document titled "New Document" and adds a paragraph of text to it. The DocumentApp.create method is used to create the document.

Exercise 5: Modify Cell Values in Google Sheets

Objective: Write a script to modify cell values in a Google Sheet.

Code Sample:

```
function updateCellValues() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  sheet.getRange("A1").setValue("Hello");
  sheet.getRange("B1").setValue("World");
}
```

Explanation:

This script updates the values of cells A1 and B1 of the active sheet to "Hello" and "World", respectively. The setValue method is used to set the value of a cell.

Exercise 6: Create an Automated Reminder in Google Calendar

Objective: Write a script to create a calendar event and set an email reminder.

Code Sample:

```
function createCalendarEvent() {
  var calendar = CalendarApp.getDefaultCalendar();
  var startTime = new Date('March 15, 2024 10:00:00');
  var endTime = new Date('March 15, 2024 11:00:00');
  var event = calendar.createEvent('Meeting with Team',
startTime, endTime);

  event.addEmailReminder(30); // 30 minutes before
}
```

Explanation:

This script creates an event in the user's default Google Calendar. It sets up a meeting titled 'Meeting with Team' at a specified date and time. An email reminder is added 30 minutes before the event starts.

Exercise 7: Access and Modify Google Drive Files

Objective: List all files in the Google Drive root folder and log their names.

Code Sample:

```
function listDriveFiles() {
  var files = DriveApp.getRootFolder().getFiles();
  while (files.hasNext()) {
    var file = files.next();
    Logger.log(file.getName());
  }
}
```

Explanation:

This script accesses the root folder of Google Drive and iterates through all files in it. It logs the name of each file found, which can be viewed in the Google Apps Script's Logger.

Exercise 8: Fetch Data from an External API

Objective: Write a script to fetch data from an external API and log the response.

Code Sample:

```
function fetchDataFromAPI() {
  var response = UrlFetchApp.fetch("https://api.example.com/data");
```

```
  Logger.log(response.getContentText());
}
```

Explanation:

This script uses the UrlFetchApp service to make a GET request to an external API. It logs the response content, which can be JSON, XML, or plain text, depending on the API.

Exercise 9: Manipulate Spreadsheet Formatting

Objective: Change the background color of the first row in a Google Sheet.

Code Sample:

```
function formatFirstRow() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange("1:1"); // First row
  range.setBackground("yellow");
}
```

Explanation:

This script gets the active sheet and selects the first row. It then changes the background color of this row to yellow, demonstrating basic cell formatting capabilities.

Exercise 10: Sync Spreadsheet Data to Google Calendar

Objective: Create Google Calendar events based on data from a Google Sheets spreadsheet.

Code Sample:

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

  for (var i = 1; i < data.length; i++) {
    var row = data[i];
    var title = row[0]; // Event title in the first
column
    var startTime = new Date(row[1]); // Start time in
the second column
    var endTime = new Date(row[2]); // End time in the
third column
    CalendarApp.getDefaultCalendar().createEvent(title,
startTime, endTime);
  }
}
```

Explanation:

This script reads data from the active sheet, assuming that each row contains an event title, start time, and end time. It loops through each row and creates corresponding events in the user's default Google Calendar. This is a basic example of how to sync spreadsheet data with Google Calendar.

Exercise 11: Generating a Report from Google Forms Responses

Objective: Create a script to generate a summary report from Google Forms responses in a Google Sheet.

Code Sample:

```
function generateFormReport() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("F
orm Responses 1");
  var data = sheet.getDataRange().getValues();
  var report = {};

  for (var i = 1; i < data.length; i++) {
    var question = data[i][1]; // Assuming question is
in the second column
    if (!report[question]) {
      report[question] = 0;
```

```
    }
    report[question]++;
  }

  Logger.log(report);
}
```

Explanation:

This script processes responses from a Google Form (stored in a Google Sheet). It tallies responses for each question and logs a summary report. This is useful for quick analysis of form data.

Exercise 12: Automating Document Creation Based on Sheet Data

Objective: Create Google Docs automatically based on rows in a Google Sheet.

Code Sample:

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

  rows.forEach(function(row, index) {
    if (index === 0) return; // Skip header row
```

```
    var doc = DocumentApp.create('Document for ' +
row[0]); // Assuming the first column has a unique
identifier
    var body = doc.getBody();
    body.appendParagraph('Data for ' + row[0]);
    // Add more content as needed
    doc.saveAndClose();
  });
}
```

Explanation:

This script iterates through rows in the active sheet and creates a Google Doc for each row. The first column is assumed to contain a unique identifier for each document. This is useful for generating personalized documents in bulk.

Exercise 13: Syncing Contacts to a Google Sheet

Objective: Import contacts from Google Contacts into a Google Sheet.

Code Sample:

```
function syncContactsToSheet() {
  var contacts = ContactsApp.getContacts();
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
```

```
  sheet.clear(); // Clear existing data
  sheet.appendRow(["Name", "Email"]); // Header row

  contacts.forEach(function(contact) {
    var name = contact.getFullName();
    var emails = contact.getEmails();
    if (emails.length > 0) {
      var email = emails[0].getAddress();
      sheet.appendRow([name, email]);
    }
  });
}
```

Explanation:

This script retrieves contacts from the user's Google Contacts and writes their names and email addresses to the active Google Sheet. This can be useful for managing contact lists or for marketing purposes.

Exercise 14: Parsing JSON Data from an API into Google Sheets

Objective: Fetch JSON data from an external API and parse it into a Google Sheet.

Code Sample:

```
function parseJSONToSheet() {
```

```
  var response =
UrlFetchApp.fetch('https://api.example.com/data');
  var json = JSON.parse(response.getContentText());
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

  json.forEach(function(item) {
    sheet.appendRow([item.name, item.value]); // Adjust
depending on JSON structure
  });
}
```

Explanation:

This script makes a GET request to an external API, parses the JSON response, and appends each item's properties as a new row in the active sheet. This is particularly useful for importing and working with dynamic data from external sources.

Exercise 15: Creating a Custom Data Dashboard in Google Sheets

Objective: Build a custom data dashboard in Google Sheets using script to update it regularly.

Code Sample:

```
function updateDashboard() {
  var dataSheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("D
ata");
  var dashboardSheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("D
ashboard");
  var data = dataSheet.getDataRange().getValues();

  // Example: Summarize data
  var total = data.reduce(function(sum, row) {
    return sum + row[1]; // Assuming data to sum is in
the second column
  }, 0);

  dashboardSheet.getRange("B2").setValue(total); //
Update a specific cell in the dashboard
}
```

Explanation:

This script reads data from a specified "Data" sheet, performs a calculation (in this case, a sum), and updates a specific cell in a "Dashboard" sheet. This is useful for creating automated, real-time dashboards in Google Sheets.

Exercise 16: Automated Email Responses Using Gmail

Objective: Write a script to automatically send a response to every email received with a specific subject.

Code Sample:

```
function autoRespondToEmails() {
  var query = 'subject:"specific subject" is:unread';
  var threads = GmailApp.search(query);
  var response = "Thank you for your email. We will get back to you shortly.";

  threads.forEach(function(thread) {
    thread.getMessages().forEach(function(message) {
      if (message.isUnread()) {
        GmailApp.sendEmail(message.getFrom(), "Re: " + message.getSubject(), response);
        message.markRead();
      }
    });
  });
}
```

Explanation:

This script searches for all unread emails with a specific subject line. For each email found, it sends a predefined response to the sender and marks the email as read. This is useful for handling common queries or out-of-office responses.

Exercise 17: Updating Google Calendar Event Details

Objective: Modify the details of upcoming events in Google Calendar.

Code Sample:

```
function updateCalendarEvents() {
  var calendar = CalendarApp.getDefaultCalendar();
  var now = new Date();
  var events = calendar.getEventsForDay(now);

  events.forEach(function(event) {
    var newTitle = "Updated: " + event.getTitle();
    event.setTitle(newTitle);
  });
}
```

Explanation:

This script retrieves all events for the current day from the user's default calendar and updates their titles with a prefix "Updated:". This can be used to dynamically change event details.

Exercise 18: Creating a Custom Google Sheets Function

Objective: Develop a custom function in Google Sheets to calculate the sum of two numbers.

Code Sample:

```
function SUM_TWO_NUMBERS(number1, number2) {
  return number1 + number2;
}
```

Explanation:

This script creates a custom function called SUM_TWO_NUMBERS that can be used directly in a Google Sheets cell, similar to built-in functions. It takes two arguments and returns their sum. In Google Sheets, you would use it as =SUM_TWO_NUMBERS(A1, B1).

Exercise 19: Fetching Weather Data and Displaying in Google Sheets

Objective: Retrieve weather data from an external API and display it in a Google Sheet.

Code Sample:

```
function fetchWeatherData() {
```

```
  var response =
UrlFetchApp.fetch("https://api.openweathermap.org/data/
2.5/weather?q=London&appid=yourAPIKey");
  var json = JSON.parse(response.getContentText());
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

  sheet.getRange("A1").setValue("Weather in London: " +
json.weather[0].main);
}
```

Explanation:

This script fetches current weather data for London from the OpenWeather API and writes the main weather description to cell A1 of the active sheet. Replace yourAPIKey with a valid API key from OpenWeather.

Exercise 20: Extracting Text from Images Using Google Drive

Objective: Use Google Drive's OCR capabilities to extract text from an image.

Code Sample:

```
function extractTextFromImage() {
  var imageId = 'yourImageFileId'; // Replace with your
image file ID in Google Drive
```

```
  var blob = DriveApp.getFileById(imageId).getBlob();
  var text = DocumentApp.create('Extracted Text')
                .getBody()

.setText(DriveApp.createFile(blob).setContentTypeFromExtension().getAs('text/plain').getDataAsString());
}
```

Explanation:

This script takes an image file from Google Drive, identified by its file ID, and uses Google Drive's built-in OCR (Optical Character Recognition) to extract text from the image. The extracted text is then written to a newly created Google Document.

Exercise 21: Generating a Table of Contents in Google Docs

Objective: Create a script to automatically generate a table of contents in a Google Document based on heading styles.

Code Sample:

```
function generateTOC() {
  var doc = DocumentApp.getActiveDocument();
  var body = doc.getBody();
  var toc = body.appendParagraph("Table of Contents");
```

```
toc.setHeading(DocumentApp.ParagraphHeading.HEADING1);

  var paragraphs = body.getParagraphs();
  paragraphs.forEach(function(paragraph) {
    if (paragraph.getHeading() ===
DocumentApp.ParagraphHeading.HEADING2) {
      var text = paragraph.getText();
      var tocEntry = body.appendParagraph(text);
      tocEntry.setLinkUrl('#' + text);
    }
  });
}
```

Explanation:

This script scans through all paragraphs in a Google Document. It identifies those with the Heading 2 style and creates a table of contents entry for each, linked to the respective headings.

Exercise 22: Batch Updating Spreadsheet Cells

Objective: Write a script to update multiple cells in Google Sheets at once, optimizing performance.

Code Sample:

```
function batchUpdateCells() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange("A1:C10");
  var values = range.getValues();

  for (var i = 0; i < values.length; i++) {
    for (var j = 0; j < values[i].length; j++) {
      values[i][j] = "Updated " + values[i][j];
    }
  }

  range.setValues(values);
}
```

Explanation:

This script reads a range of cells, modifies each cell's value, and then writes them back in one batch operation. This approach is more efficient than updating each cell individually.

Exercise 23: Automatically Archiving Emails in Gmail

Objective: Develop a script to automatically archive emails older than a specified number of days.

Code Sample:

```
function archiveOldEmails() {
  var daysOld = 30;
  var date = new Date();
  date.setDate(date.getDate() - daysOld);
  var threads = GmailApp.search('before:' +
Utilities.formatDate(date, Session.getScriptTimeZone(),
'yyyy/MM/dd'));

  threads.forEach(function(thread) {
    thread.moveToArchive();
  });
}
```

Explanation:

This script calculates the date that is 30 days before the current date. It then searches for and archives all email threads that are older than this date.

Exercise 24: Creating a Google Sheets Chart from Data

Objective: Use Apps Script to create a chart in Google Sheets based on existing data.

Code Sample:

```
function createChart() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getRange("A1:B10"); // Assuming
this range has the data
  var chartBuilder = sheet.newChart();

  chartBuilder.addRange(range)
      .setChartType(Charts.ChartType.BAR)
      .setPosition(5, 5, 0, 0);

  sheet.insertChart(chartBuilder.build());
}
```

Explanation:

This script creates a bar chart based on the data in the range A1:B10. The newChart method is used to construct the chart, which is then inserted into the sheet.

Exercise 25: Scheduling Calendar Events from Spreadsheet Data

Objective: Create a script to read event data from a spreadsheet and schedule them in Google Calendar.

Code Sample:

```
function scheduleEventsFromSheet() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("E
vents");
  var rows = sheet.getDataRange().getValues();
  var calendar = CalendarApp.getDefaultCalendar();

  rows.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);
  });
}
```

Explanation:

This script accesses a sheet named "Events", where each row contains an event title, start time, and end time. It iterates through each row and creates corresponding events in the user's default calendar.