1) Add a Custom Menu on Open (Sheets)

Does: Adds a menu to your spreadsheet UI.
Use when: You want easy buttons for scripts.

```
function onOpen() {
  SpreadsheetApp.getUi()
    .createMenu('Tools')
    .addItem('Say Hello', 'sayHello')
    .addSeparator()
    .addItem('Run Cleanup', 'cleanup')
    .addToUi();
}

function sayHello() {
  SpreadsheetApp.getUi().alert('Hello from Apps Script!');
}

function cleanup() {
  SpreadsheetApp.getUi().alert('Cleanup placeholder.');
}
```

---

2) Show a Sidebar (Sheets)

Does: Opens a sidebar with HTML UI.
Use when: You want a mini-app inside Sheets.

```
function showSidebar() {
  const html = HtmlService
    .createHtmlOutput('<h3>Sidebar</h3><p>Hello!</p>')
    .setTitle('My Sidebar');
  SpreadsheetApp.getUi().showSidebar(html);
}
```

---

3) Show a Modal Dialog (Sheets)

Does: Shows a pop-up dialog with HTML content.
Use when: Confirmations, forms, help screens.

```
function showDialog() {
  const html = HtmlService
    .createHtmlOutput('<h2>Dialog</h2><p>This is a modal.</p>')
    .setWidth(400)
    .setHeight(200);
  SpreadsheetApp.getUi().showModalDialog(html, 'My Dialog');
}
```

---

4) Read a Single Cell Value (Sheets)

Does: Reads a value from A1.
Use when: You need input from a specific cell.

```
function readCellA1() {
  const sh = SpreadsheetApp.getActiveSheet();
  const value = sh.getRange('A1').getValue();
```

```
  Logger.log(value);
}
```

---

5) Write a Value to a Cell (Sheets)

Does: Writes text into B2.
Use when: Output results into the sheet.

```
function writeToB2() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange('B2').setValue('Done!');
}
```

---

6) Append a Row to a Sheet

Does: Adds a new row at the bottom.
Use when: Logging, form-like data entry.

```
function appendRow() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.appendRow([new Date(), 'Event',
Session.getActiveUser().getEmail()]);
}
```

---

7) Read a Whole Data Range (Fast)

Does: Reads all values in one call.
Use when: Performance matters.

```
function readAllData() {
  const sh = SpreadsheetApp.getActiveSheet();
  const data = sh.getDataRange().getValues(); // 2D array
  Logger.log(`Rows: ${data.length}, Cols: ${data[0].length}`);
}
```

---

8) Write a Whole Block (Fast)

Does: Writes a 2D array at once.
Use when: Output tables efficiently.

```
function writeBlock() {
  const sh = SpreadsheetApp.getActiveSheet();
  const values = [
    ['Name', 'Score'],
    ['Ava', 95],
    ['Noah', 88],
  ];
  sh.getRange(1, 1, values.length,
values[0].length).setValues(values);
}
```

---

9) Clear Only Contents (Keep Formatting)

Does: Clears cell values but keeps formatting.
Use when: Resetting input sheets.

```
function clearContentsOnly() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange('A2:Z').clearContent();
}
```

---

10) Clear Formatting Only

Does: Removes formatting but keeps values.
Use when: Normalize messy sheets.

```
function clearFormattingOnly() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getDataRange().clearFormat();
```

```
}
```

---

11) Find Duplicate Values in a Column

Does: Lists duplicates from column A.
Use when: Data cleaning.

```
function findDuplicatesInColA() {
  const sh = SpreadsheetApp.getActiveSheet();
  const values =
sh.getRange('A2:A').getValues().flat().filter(String);
  const seen = new Set();
  const dupes = new Set();

  values.forEach(v => {
    const key = String(v).trim();
    if (seen.has(key)) dupes.add(key);
    else seen.add(key);
  });

  Logger.log([...dupes]);
}
```

---

12) Highlight Duplicates in Column A

Does: Turns duplicate cells red.
Use when: Visual duplicate detection.

```
function highlightDuplicatesInColA() {
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getRange('A2:A' + sh.getLastRow());
  const values = range.getValues().flat();

  const counts = values.reduce((m, v) => {
    const k = String(v).trim();
    if (!k) return m;
```

```
    m[k] = (m[k] || 0) + 1;
    return m;
  }, {});

  const backgrounds = values.map(v => {
    const k = String(v).trim();
    if (!k) return ['#ffffff'];
    return [counts[k] > 1 ? '#ffcccc' : '#ffffff'];
  });

  range.setBackgrounds(backgrounds);
}
```

---

13) Sort a Range by a Column

Does: Sorts A2:D by column 2 ascending.
Use when: Auto-sorting tables.

```
function sortTable() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange('A2:D' + sh.getLastRow()).sort({ column: 2, ascending:
true });
}
```

---

14) Filter Rows by Condition and Output to Another Sheet

Does: Copies rows where Score >= 90.
Use when: Create “top performers” view.

```
function filterScoresToSheet() {
  const ss = SpreadsheetApp.getActive();
  const src = ss.getSheetByName('Sheet1') || ss.getActiveSheet();
  const dst = ss.getSheetByName('Top') || ss.insertSheet('Top');

  const data = src.getDataRange().getValues();
  const header = data[0];
```

```
  const rows = data.slice(1).filter(r => Number(r[1]) >= 90); // score
in col B

  dst.clear();
  dst.getRange(1, 1, 1, header.length).setValues([header]);
  if (rows.length) dst.getRange(2, 1, rows.length,
header.length).setValues(rows);
}
```

---

15) Create a New Sheet if Missing

Does: Ensures a sheet exists.
Use when: Scripts need predictable tabs.

```
function ensureSheet(name) {
  const ss = SpreadsheetApp.getActive();
  return ss.getSheetByName(name) || ss.insertSheet(name);
}

function demoEnsureSheet() {
  const sh = ensureSheet('Logs');
  sh.appendRow(['Created/Found', new Date()]);
}
```

---

16) Protect a Range (Sheets)

Does: Protects A1:D1 from edits.
Use when: Lock headers.

```
function protectHeaderRow() {
  const sh = SpreadsheetApp.getActiveSheet();
  const protection =
sh.getRange('A1:D1').protect().setDescription('Protect header');
  protection.removeEditors(protection.getEditors()); // keep owner
only
}
```

---

17) Data Validation Dropdown (Sheets)

Does: Creates a dropdown in B2:B20.
Use when: Controlled inputs.

```
function addDropdown() {
  const sh = SpreadsheetApp.getActiveSheet();
  const rule = SpreadsheetApp.newDataValidation()
    .requireValueInList(['Todo', 'Doing', 'Done'], true)
    .setAllowInvalid(false)
    .build();

  sh.getRange('B2:B20').setDataValidation(rule);
}
```

---

18) Timestamp When a Cell Is Edited (Simple)

Does: When column A changes, writes timestamp into column B.
Use when: Track edits.

```
function onEdit(e) {
  const range = e.range;
  const sh = range.getSheet();
  if (sh.getName() !== 'Sheet1') return;

  if (range.getColumn() === 1 && range.getRow() > 1) {
    sh.getRange(range.getRow(), 2).setValue(new Date());
  }
}
```

---

19) Send Email (Gmail)

Does: Sends a basic email.
Use when: Notifications.

```
function sendEmailDemo() {
  GmailApp.sendEmail(
    'someone@example.com',
    'Apps Script Test',
    'Hello! This is a test email from Apps Script.'
  );
}
```

---

20) Email a Sheet as PDF Attachment

Does: Exports current sheet as PDF and emails it.
Use when: Weekly reports.

```
function emailActiveSheetAsPdf() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();
  const url = ss.getUrl().replace(/edit$/, '') +
    'export?format=pdf&gid=' + sh.getSheetId();

  const token = ScriptApp.getOAuthToken();
  const blob = UrlFetchApp.fetch(url, {
    headers: { Authorization: 'Bearer ' + token }
  }).getBlob().setName(sh.getName() + '.pdf');

  GmailApp.sendEmail('someone@example.com', 'Sheet PDF', 'See
attached.', {
    attachments: [blob]
  });
}
```

---

21) Create a Google Doc and Write Text

Does: Creates a doc and fills it.
Use when: Generate reports.

```
function createDocReport() {
  const doc = DocumentApp.create('My Report');
  const body = doc.getBody();
  body.appendParagraph('Report
Title').setHeading(DocumentApp.ParagraphHeading.HEADING1);
  body.appendParagraph('Generated: ' + new Date());
  Logger.log(doc.getUrl());
}
```

---

22) Replace Text in a Google Doc

Does: Finds and replaces text in active doc.
Use when: Template filling.

```
function replaceTextInActiveDoc() {
  const doc = DocumentApp.getActiveDocument();
  doc.getBody().replaceText('{{NAME}}', 'Lars');
}
```

---

23) Create a Doc from a Template File

Does: Copies a template doc, then replaces placeholders.
Use when: Document automation.

```
function createFromTemplate() {
  const templateId = 'PASTE_TEMPLATE_DOC_ID';
  const folderId = 'PASTE_FOLDER_ID';

  const copy = DriveApp.getFileById(templateId).makeCopy('Filled
Template', DriveApp.getFolderById(folderId));
  const doc = DocumentApp.openById(copy.getId());

  doc.getBody()
```

```
    .replaceText('{{DATE}}', Utilities.formatDate(new Date(),
Session.getScriptTimeZone(), 'yyyy-MM-dd'))
    .replaceText('{{TITLE}}', 'Weekly Update');

  doc.saveAndClose();
  Logger.log(doc.getUrl());
}
```

---

24) Export a Doc as PDF (Blob)

Does: Gets PDF blob from a Doc file.
Use when: Attach PDF in email.

```
function docToPdfBlob(docId) {
  const file = DriveApp.getFileById(docId);
  return file.getBlob().getAs(MimeType.PDF).setName(file.getName() +
'.pdf');
}
```

---

25) Create a Calendar Event

Does: Adds an event to your primary calendar.
Use when: Automated scheduling.

```
function createCalendarEvent() {
  const cal = CalendarApp.getDefaultCalendar();
  const start = new Date();
  const end = new Date(start.getTime() + 60 * 60 * 1000);
  cal.createEvent('Apps Script Event', start, end, { description:
'Created by script' });
}
```

---

26) List Upcoming Calendar Events

Does: Logs next 10 events.
Use when: Dashboards, summaries.

```
function listUpcomingEvents() {
  const cal = CalendarApp.getDefaultCalendar();
  const now = new Date();
  const future = new Date(now.getTime() + 7 * 24 * 60 * 60 * 1000);
  const events = cal.getEvents(now, future).slice(0, 10);

  events.forEach(e => Logger.log(`${e.getTitle()} –
${e.getStartTime()}`));
}
```

---

27) Create a Form Programmatically

Does: Creates a Google Form with questions.
Use when: Bulk form creation.

```
function createForm() {
  const form = FormApp.create('Feedback Form');
  form.addTextItem().setTitle('Name').setRequired(true);
  form.addMultipleChoiceItem()
    .setTitle('Rate the session')
    .setChoiceValues(['1', '2', '3', '4', '5'])
    .setRequired(true);

  Logger.log(form.getEditUrl());
}
```

---

28) Read Latest Form Responses into Sheet

Does: Grabs last response from a form linked to spreadsheet.
Use when: Post-process form results.

```
function logLatestFormResponse() {
  const form = FormApp.getActiveForm();
```

```
  const responses = form.getResponses();
  if (!responses.length) return;

  const last = responses[responses.length - 1];
  const items = last.getItemResponses().map(r =>
`${r.getItem().getTitle()}: ${r.getResponse()}`);
  Logger.log(items.join('\n'));
}
```

---

29) Create a Drive Folder (If Missing)

Does: Creates folder if not already there.
Use when: Organize outputs.

```
function getOrCreateFolder(name) {
  const folders = DriveApp.getFoldersByName(name);
  return folders.hasNext() ? folders.next() :
DriveApp.createFolder(name);
}

function demoFolder() {
  const f = getOrCreateFolder('Apps Script Outputs');
  Logger.log(f.getUrl());
}
```

---

30) Move a File to a Folder

Does: Moves file by ID into a folder.
Use when: File organization automation.

```
function moveFileToFolder(fileId, folderId) {
  const file = DriveApp.getFileById(fileId);
  const folder = DriveApp.getFolderById(folderId);
  folder.addFile(file);

  // Optional: remove from root
```

```
  DriveApp.getRootFolder().removeFile(file);
}
```

---

31) Create a CSV File in Drive

Does: Generates a CSV file from values.
Use when: Export for other systems.

```
function createCsvFile() {
  const rows = [
    ['Name', 'Score'],
    ['Ava', 95],
    ['Noah', 88],
  ];
  const csv = rows.map(r => r.map(String).join(',')).join('\n');
  const file = DriveApp.createFile('scores.csv', csv, MimeType.CSV);
  Logger.log(file.getUrl());
}
```

---

32) Fetch JSON from an API (UrlFetch)

Does: Calls an API and parses JSON.
Use when: Integrations.

```
function fetchJsonDemo() {
  const res = UrlFetchApp.fetch('https://api.publicapis.org/entries');
  const json = JSON.parse(res.getContentText());
  Logger.log(`Count: ${json.count}`);
}
```

---

33) POST JSON to an API

Does: Sends a JSON payload.
Use when: Webhooks.

```
function postJsonDemo() {
  const url = 'https://httpbin.org/post';
  const payload = { event: 'test', time: new Date().toISOString() };

  const res = UrlFetchApp.fetch(url, {
    method: 'post',
    contentType: 'application/json',
    payload: JSON.stringify(payload),
    muteHttpExceptions: true
  });

  Logger.log(res.getResponseCode());
  Logger.log(res.getContentText());
}
```

---

34) Retry Fetch with Backoff

Does: Retries a flaky request 3 times.
Use when: APIs time out.

```
function fetchWithRetry(url) {
  let lastErr;
  for (let i = 0; i < 3; i++) {
    try {
      return UrlFetchApp.fetch(url, { muteHttpExceptions: true });
    } catch (err) {
      lastErr = err;
      Utilities.sleep((i + 1) * 1000);
    }
  }
  throw lastErr;
}

function demoRetry() {
  const res = fetchWithRetry('https://example.com');
  Logger.log(res.getResponseCode());
}
```

---

35) Store a Setting in Script Properties

Does: Saves key/value config.
Use when: Avoid hardcoding.

```
function setConfig() {
  PropertiesService.getScriptProperties().setProperty('API_KEY',
'demo-key');
}

function getConfig() {
  const key =
PropertiesService.getScriptProperties().getProperty('API_KEY');
  Logger.log(key);
}
```

---

36) Per-User Settings with User Properties

Does: Stores preferences per user.
Use when: Multiple users use same script.

```
function setUserPref() {
  PropertiesService.getUserProperties().setProperty('theme', 'dark');
}

function getUserPref() {

Logger.log(PropertiesService.getUserProperties().getProperty('theme'))
;
}
```

---

37) Cache Expensive Results (CacheService)

Does: Caches output for 5 minutes.
Use when: Repeated calls, speed.

```
function cachedHello() {
  const cache = CacheService.getScriptCache();
  const key = 'hello';
  const cached = cache.get(key);
  if (cached) return cached;

  const value = 'Hello at ' + new Date().toISOString();
  cache.put(key, value, 300);
  return value;
}
```

---

38) Create a Time-Driven Trigger

Does: Runs a function hourly.
Use when: Scheduled automations.

```
function createHourlyTrigger() {
  ScriptApp.newTrigger('hourlyJob')
    .timeBased()
    .everyHours(1)
    .create();
}

function hourlyJob() {
  Logger.log('Hourly job ran at ' + new Date());
}
```

---

39) Delete All Triggers for a Function

Does: Cleans up triggers.
Use when: Reset schedules.

```
function deleteTriggersFor(functionName) {
```

```
  ScriptApp.getProjectTriggers().forEach(t => {
    if (t.getHandlerFunction() === functionName)
ScriptApp.deleteTrigger(t);
  });
}

function demoDeleteHourly() {
  deleteTriggersFor('hourlyJob');
}
```

---

40) Log to a “Logs” Sheet

Does: Writes timestamped logs.
Use when: You want logs visible in Sheets.

```
function logToSheet(message) {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getSheetByName('Logs') || ss.insertSheet('Logs');
  sh.appendRow([new Date(), message]);
}

function demoLog() {
  logToSheet('Script started');
}
```

---

41) Create a Simple Web App (doGet)

Does: Returns HTML from a URL endpoint.
Use when: Basic web app / landing page.

```
function doGet() {
  return HtmlService.createHtmlOutput('<h1>Hello Web App</h1><p>It
works.</p>');
}
```

---

42) Web App JSON Endpoint

Does: Returns JSON.
Use when: API for your own apps.

```
function doGet() {
  const data = { ok: true, time: new Date().toISOString() };
  return ContentService
    .createTextOutput(JSON.stringify(data))
    .setMimeType(ContentService.MimeType.JSON);
}
```

---

43) Parse Query Params in doGet

Does: Reads `?name=Lars` from URL.
Use when: Personalized responses.

```
function doGet(e) {
  const name = (e && e.parameter && e.parameter.name) ?
e.parameter.name : 'friend';
  return HtmlService.createHtmlOutput(`<h2>Hello, ${name}!</h2>`);
}
```

---

44) Simple HTML Form + doPost

Does: Handles form submission.
Use when: Collect data into Sheets.

```
function doGet() {
  const html = `
    <form method="post">
      <label>Name <input name="name"/></label>
      <button type="submit">Send</button>
    </form>`;
  return HtmlService.createHtmlOutput(html);
}
```

```
function doPost(e) {
  const name = e.parameter.name || '';
  return HtmlService.createHtmlOutput(`<p>Thanks, ${name}!</p>`);
}
```

---

45) Create a Named Range

Does: Names A1:B10 as “MyRange”.
Use when: Stable references.

```
function createNamedRange() {
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getRange('A1:B10');
  SpreadsheetApp.getActive().setNamedRange('MyRange', range);
}
```

---

46) Read a Named Range

Does: Reads from “MyRange”.
Use when: Templates rely on names.

```
function readNamedRange() {
  const range = SpreadsheetApp.getActive().getRangeByName('MyRange');
  if (!range) throw new Error('Named range not found');
  Logger.log(range.getValues());
}
```

---

47) Add Conditional Formatting Rule

Does: Highlights values > 100 in C2:C.
Use when: Visual thresholds.

```
function addConditionalFormatting() {
```

```
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getRange('C2:C' + sh.getLastRow());

  const rule = SpreadsheetApp.newConditionalFormatRule()
    .whenNumberGreaterThan(100)
    .setBackground('#fff2cc')
    .setRanges([range])
    .build();

  const rules = sh.getConditionalFormatRules();
  rules.push(rule);
  sh.setConditionalFormatRules(rules);
}
```

---

48) Create a Pivot Table (Basic)

Does: Creates a pivot in a new sheet from data in A1:D.
Use when: Summaries.

```
function createPivot() {
  const ss = SpreadsheetApp.getActive();
  const src = ss.getActiveSheet();
  const dataRange = src.getRange(1, 1, src.getLastRow(),
src.getLastColumn());

  const pivotSheet = ss.getSheetByName('Pivot') ||
ss.insertSheet('Pivot');
  pivotSheet.clear();

  const pivotAnchor = pivotSheet.getRange('A1');
  const pivot = pivotAnchor.createPivotTable(dataRange);

  // Example: row group by column 1, sum column 2
  pivot.addRowGroup(1);
  pivot.addPivotValue(2,
SpreadsheetApp.PivotTableSummarizeFunction.SUM);
}
```

---

49) Remove Empty Rows (Sheets)

Does: Deletes blank rows based on column A being empty.
Use when: Cleaning exports.

```
function deleteEmptyRowsBasedOnColA() {
  const sh = SpreadsheetApp.getActiveSheet();
  const lastRow = sh.getLastRow();
  if (lastRow < 2) return;

  const values = sh.getRange(2, 1, lastRow - 1, 1).getValues().flat();
  for (let i = values.length - 1; i >= 0; i--) {
    if (!String(values[i]).trim()) sh.deleteRow(i + 2);
  }
}
```

---

50) Create a “Run Report” Summary (Sheets)

Does: Generates a mini report of sheet stats.
Use when: Quick diagnostics.

```
function sheetReport() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();

  const report = [
    ['Spreadsheet', ss.getName()],
    ['Sheet', sh.getName()],
    ['Last Row', sh.getLastRow()],
    ['Last Column', sh.getLastColumn()],
    ['URL', ss.getUrl()],
    ['Generated', new Date()],
  ];

  const out = ss.getSheetByName('Report') || ss.insertSheet('Report');
```

```
  out.clear();
  out.getRange(1, 1, report.length, 2).setValues(report);
  out.autoResizeColumns(1, 2);
}
```
