101) Custom menu

- **What it does:** Adds a new top-level menu to the Google Sheets UI.
- **How it works:** `SpreadsheetApp.getUi().createMenu()` builds a menu and maps menu items to function names.
- **When to use:** Turn scripts into click-to-run tools for non-technical users.
- **Gotchas:** Must be **Sheets-bound** to show menus. Usually paired with `onOpen()`.

102) Sidebar UI

- **What it does:** Opens a sidebar panel inside Sheets using HTML.

- **How it works:** `HtmlService.createHtmlOutput()` creates a small HTML document and `showSidebar()` renders it.
- **When to use:** Forms, dashboards, buttons, user prompts.
- **Gotchas:** Sidebar HTML is sandboxed; use `google.script.run` to call server functions.

103) Find & replace (entire sheet)

- **What it does:** Replaces a text value everywhere in the active sheet.
- **How it works:** `createTextFinder()` scans cells and `replaceAllWith()` replaces matches.
- **When to use:** Bulk cleanup after imports.
- **Gotchas:** Works on displayed values; formatting/formulas may still produce text.

104) Freeze row + column

- **What it does:** Freezes header row and first column.
- **How it works:** `setFrozenRows(1)` and `setFrozenColumns(1)`.
- **When to use:** Any tracker/report with headers.
- **Gotchas:** Freezing too many rows can annoy users—keep it minimal.

105) Auto-resize columns

- **What it does:** Auto-sizes columns based on content.
- **How it works:** `autoResizeColumns(startCol, numCols)` measures rendered widths.
- **When to use:** After generating reports or pasting data.

- **Gotchas:** Large sheets can take time.

106) Dropdown validation

- **What it does:** Adds a dropdown list to a range and blocks invalid input.
- **How it works:** `newDataValidation().requireValueInList(...).setAllowInvalid(false)`.
- **When to use:** Standardize statuses/categories.
- **Gotchas:** If you later change allowed values, you must reset validation.

107) Add checkboxes

- **What it does:** Converts range cells into checkboxes.
- **How it works:** `insertCheckboxes()` changes cell data validation to checkbox type.
- **When to use:** Task completion, approvals.
- **Gotchas:** Checkboxes store TRUE/FALSE, which affects formulas.

108) Protect sheet except input range

- **What it does:** Locks most of the sheet but leaves a specified range editable.
- **How it works:** `protect()` creates a protection object; `setUnprotectedRanges()` whitelists editable zones.
- **When to use:** Shared templates with safe input areas.
- **Gotchas:** Protection permissions can vary by user; test with another account.

109) Create filter

- **What it does:** Adds a filter to the data range if none exists.
- **How it works:** Checks `getFilter()` then `createFilter()`.
- **When to use:** Quick usability improvement for reports.
- **Gotchas:** Filters can hide rows—other scripts must handle hidden rows properly.

110) Multi-column sort

- **What it does:** Sorts dataset by multiple keys.
- **How it works:** Range `.sort([{column:2},{column:1}])` applies stable sort rules.
- **When to use:** Normalize ordering for exports/reports.
- **Gotchas:** Sorting affects row position—avoid if other systems rely on row numbers.

111) Split text into columns

- **What it does:** Splits values from one column into multiple columns.
- **How it works:** Reads A, `.split('|')`, finds max number of parts, pads, writes to B onward.
- **When to use:** Parsing imported data like “first|last|email”.
- **Gotchas:** Overwrites destination cells; ensure target columns are empty.

112) Cleanup trim + whitespace

- **What it does:** Trims edges and collapses repeated spaces inside strings.
- **How it works:** For string cells: `.trim().replace(/\s+/g,' ')`.
- **When to use:** Dirty data from copy/paste, form fills, imports.
- **Gotchas:** Doesn’t change non-string cells (numbers/dates).

113) Title Case column

- **What it does:** Converts text to Title Case.
- **How it works:** Lowercases then capitalizes word boundaries using regex.
- **When to use:** Names, categories, titles.
- **Gotchas:** “McDonald”/“iPhone” style names won’t be perfect automatically.

114) Timestamp when status changes (onEdit)

- **What it does:** Writes a timestamp when a cell in a certain column is edited.
- **How it works:** Trigger function receives event object `e`, checks edited column/row, writes new Date().
- **When to use:** Audit trails / “last updated” tracking.
- **Gotchas:** Simple triggers have limits; installable trigger is more reliable.

115) Copy visible rows only

- **What it does:** Copies rows that aren’t hidden by filter or user.
- **How it works:** Iterates rows, checks `isRowHiddenByFilter()` and `isRowHiddenByUser()`.
- **When to use:** Export filtered results.
- **Gotchas:** Requires reading all rows; large sheets = slower.

116) Create pivot table

- **What it does:** Builds a pivot table automatically.

- **How it works:** `createPivotTable(sourceRange)`, `addRowGroup`, `addPivotValue`.
- **When to use:** Auto analytics dashboards.
- **Gotchas:** Pivot config is basic; complex pivots need more setup.

117) Conditional format top 10%

- **What it does:** Highlights values above the 90th percentile.
- **How it works:** Adds a rule using a formula: `B2>=PERCENTILE(...)`.
- **When to use:** Outperformance highlighting.
- **Gotchas:** Percentile recalculates; ensure column ranges are correct.

118) Named range for data table

- **What it does:** Creates a named range pointing at current `getDataRange()`.
- **How it works:** `setNamedRange('DATA_TABLE', range)`.
- **When to use:** Stable reference for scripts and formulas.
- **Gotchas:** If your data grows/shrinks, rerun to update.

119) Convert formulas to values

- **What it does:** Replaces formulas with their current evaluated values.
- **How it works:** `copyTo` with `{contentsOnly:true}`.
- **When to use:** Freeze snapshot for exports.
- **Gotchas:** This is irreversible unless you undo immediately.

120) Hyperlink helper

- **What it does:** Turns raw URLs into clickable links with label text.
- **How it works:** Writes `=HYPERLINK("url","Open")` formulas into column B.
- **When to use:** Cleaner presentation.
- **Gotchas:** If URL contains quotes, needs escaping.

121) Generate UUIDs

- **What it does:** Adds unique IDs to each row.
- **How it works:** `Utilities.getUuid()` per row.
- **When to use:** Stable row identifiers for syncing.
- **Gotchas:** If you rerun it, IDs change unless you only fill blanks.

122) Missing values report

- **What it does:** Scans a table and outputs blanks with row/col/header info.
- **How it works:** Nested loop checks for `''` or `null`, writes rows to a report sheet.
- **When to use:** Data QA / validation.
- **Gotchas:** Uses header row to label columns—must have headers.

123) Sync two sheets by ID (left join)

- **What it does:** Combines SheetA + SheetB by matching ID in column A.
- **How it works:** Builds a Map of SheetB rows keyed by ID, then appends matching fields.
- **When to use:** Enrich dataset with lookup table data.

- **Gotchas:** IDs must match exactly after trimming; duplicates in SheetB will overwrite.

124) Insert a chart

- **What it does:** Creates and inserts a chart in the sheet.
- **How it works:** `newChart()`, `addRange()`, `setPosition()`, `insertChart()`.
- **When to use:** Auto reporting.
- **Gotchas:** Chart types vary; data layout matters.

125) Remove all charts

- **What it does:** Deletes all chart objects.
- **How it works:** `getCharts()` returns chart objects; `removeChart()` removes each.
- **When to use:** Reset dashboards.
- **Gotchas:** No “undo” after script finishes (unless user manually undoes quickly).

126) Duplicate sheet with timestamp

- **What it does:** Makes a copy of the current sheet as a snapshot.
- **How it works:** `copyTo(ss)` then rename using formatted timestamp.
- **When to use:** Backups before transformations.
- **Gotchas:** Duplicate inherits protections; rename collisions possible.

127) Append daily log

- **What it does:** Appends a log entry row.

- **How it works:** `appendRow([date,email,message])`.
- **When to use:** Track script runs or user activity.
- **Gotchas:** AppendRow can be slow at very large scales.

128) Soft lock with CacheService

- **What it does:** Prevents two runs from overlapping.
- **How it works:** Writes a cache key for 10 minutes; blocks if key exists.
- **When to use:** Avoid double execution from button spam/triggers.
- **Gotchas:** Cache is best-effort; use `LockService` for stronger locking.

129) Run report sheet

- **What it does:** Creates a new sheet that logs run time.
- **How it works:** `insertSheet()` then writes values.
- **When to use:** Debug / audit runs.
- **Gotchas:** Will create many sheets if run often.

130) Currency strings to numbers

- **What it does:** Strips non-numeric characters and converts to Number.
- **How it works:** Regex removes symbols/commas, `Number()` converts.
- **When to use:** Imported financial data.
- **Gotchas:** Different locales use commas/periods differently.

131) Remove duplicates by column

- **What it does:** Removes duplicates based on selected columns (column A).
- **How it works:** `removeDuplicates([1])`.
- **When to use:** Deduping contact lists or IDs.
- **Gotchas:** Only works on `getDataRange()` you apply it to.

132) Alternating row banding

- **What it does:** Applies alternating row shading.
- **How it works:** `applyRowBanding()`.
- **When to use:** Readability.
- **Gotchas:** Can conflict with custom formatting rules.

133) Top N extraction

- **What it does:** Builds a “Top 10” sheet based on column B.
- **How it works:** Reads all, sorts descending, slices 10, writes to new tab.
- **When to use:** Leaderboards or performance snapshots.
- **Gotchas:** Sorting treats blanks as 0; ensure numeric types.

134) Outlier detection (z-score)

- **What it does:** Flags statistically extreme values.
- **How it works:** Computes mean + std dev; z = (x-mean)/sd.
- **When to use:** Data QA / anomaly detection.
- **Gotchas:** Requires enough values; sd=0 means no variation.

135) Parse dates

- **What it does:** Converts date strings into real Date objects.
- **How it works:** `new Date(v)` and checks `isNaN(time)`.
- **When to use:** Fixing imported dates.
- **Gotchas:** Date parsing depends on format; ambiguous strings can parse wrong.

136) Monthly summary

- **What it does:** Aggregates counts per month.
- **How it works:** Formats each date as `yyyy-MM` and counts in an object map.
- **When to use:** Trend reporting.
- **Gotchas:** Uses spreadsheet timezone—important for near-midnight times.

137) Import CSV string

- **What it does:** Parses CSV text into a 2D array and writes to sheet.
- **How it works:** `Utilities.parseCsv(csv)` returns rows.
- **When to use:** Simple demo imports / API CSV data.
- **Gotchas:** For large CSVs, use Drive file + streaming patterns.

138) Export to JSON

- **What it does:** Converts table rows into JSON objects keyed by header row.
- **How it works:** Creates header array, uses `Object.fromEntries`.
- **When to use:** API payload creation, backups, integrations.

- **Gotchas:** Duplicate header names will overwrite keys.

139) Notes from column E

- **What it does:** Copies values from a “notes column” into cell notes.
- **How it works:** `setNotes()` expects a 2D array of strings.
- **When to use:** Keep explanations/comments attached to cells.
- **Gotchas:** Notes are not the same as comments.

140) Clear content, keep formatting

- **What it does:** Clears values but preserves formatting and layout.
- **How it works:** `clearContent()` doesn’t touch formatting.
- **When to use:** Reset template sheets for reuse.
- **Gotchas:** Doesn’t remove checkboxes/data validation—only values.

141) Create folder + log URL

What it does:
Creates a new Drive folder and logs the folder URL.

How it works:
`DriveApp.createFolder(name)` returns a Folder object → `getUrl()` prints share link.

When to use:

- Automated project setup
- Creating “run output” folders for exports/backups

Gotchas:

- Requires Drive authorization

- Folder name collisions are allowed (Drive supports duplicates)

---

142) List files in a folder

What it does:
Logs the names of all files inside a specific folder.

How it works:
`getFolderById()` → `getFiles()` iterator → loop `hasNext()/next()`.

When to use:

- Auditing folder contents
- Building a report/inventory

Gotchas:

- You must replace `PASTE_FOLDER_ID`
- Doesn't include subfolders (only files directly inside)

---

143) Copy a file into a folder

What it does:
Creates a duplicate of a file and places the copy in a target folder.

How it works:
`getFileById().makeCopy(destinationFolder)` returns the new file.

When to use:

- Templating files (copy a template into a job folder)
- Creating backups before edits

Gotchas:

- Replace both file and folder IDs
- Sharing/permissions on the copy can differ from original depending on domain policies

---

144) Move file to folder

What it does:
Moves a file into a target folder and removes it from all existing parent folders.

How it works:

- `target.addFile(file)`
- Iterates `file.getParents()` and removes file from each.

When to use:

- Inbox→Processed workflows
- Organizing Drive with automations

Gotchas:

- If the file has multiple parents, this removes all of them (true move)
- Requires Drive authorization

---

145) Search Drive by name

What it does:
Searches Drive for files whose names contain a substring (example: “Report”).

How it works:
`DriveApp.searchFiles(query)` uses Drive query syntax.

When to use:

- Finding generated reports
- Automations based on naming conventions

Gotchas:

- Queries can match lots of files—limit results for performance
- Search doesn't automatically scope to a folder unless you include folder criteria (more advanced)

---

146) Create a text file in folder

What it does:
Creates a `.txt` file inside a folder with provided contents.

How it works:
`folder.createFile(name, content, mimeType)`.

When to use:

- Writing logs, summaries, exported metadata
- Creating simple "handoff notes" automatically

Gotchas:

- Replace folder ID
- Re-running may create duplicates unless you search/update instead

---

147) Overwrite text file contents

What it does:
Updates an existing text file to new content.

How it works:
`DriveApp.getFileById(fileId).setContent(text)` replaces contents.

When to use:

- “Latest status” file that always stays current
- Rolling reports

Gotchas:

- Works best on Google Docs text files / plain text content files
- Replace file ID

---

148) Make file public-view (link sharing)

What it does:
Sets a file to “Anyone with link can view.”

How it works:
`setSharing(Access.ANYONE_WITH_LINK, Permission.VIEW)`.

When to use:

- Sharing generated exports to external people
- Publishing static HTML/PDF reports

Gotchas:

- Some Workspace domains block public link sharing
- Be careful with sensitive data

---

149) Create manifest.json (name → id)

What it does:
Builds a JSON index of every file in a folder mapping filename to file ID, then saves it as `manifest.json`.

How it works:

- Iterates files
- Builds object `{name:id}`
- Writes blob file with `application/json`

When to use:

- Creating lookup tables for automation pipelines
- Handing off folder contents to other scripts/tools

Gotchas:

- Duplicate filenames will overwrite keys in the manifest
- Only includes direct files, not recursive subfolders

---

150) Zip all files in a folder

What it does:
Creates a ZIP archive containing all files in a folder and saves the zip back into that folder.

How it works:

- Collects each file's blob
- `Utilities.zip(blobs, 'bundle.zip')`
- Creates zip file in Drive

When to use:

- Packaging deliverables
- Creating download bundles for clients/users

Gotchas:

- Apps Script size limits apply (very large folders can fail)
- Google-native files (Docs/Sheets) convert to export blobs depending on blob retrieval behavior

---

151) Trash empty subfolders

What it does:
Finds empty subfolders (no files and no nested folders) and trashes them.

How it works:
Iterates `root.getFolders()` and checks `getFiles().hasNext()` and `getFolders().hasNext()`.

When to use:

- Cleaning up auto-generated project folders
- Removing abandoned directories

Gotchas:

- One-level deep only
- Trashing is reversible, but still destructive—be cautious

---

152) Rename files by replacement

What it does:
Renames all files in a folder by replacing a substring (Draft → Final).

How it works:
Loops files and runs `setName(f.getName().replace(...))`.

When to use:

- Standardizing naming conventions
- Finalizing batch deliverables

Gotchas:

- Replace can unintentionally rename parts of names you didn't mean
- Doesn't handle duplicate naming collisions gracefully (Drive allows duplicates though)

---

153) Shortcut creation note

What it does:
Logs guidance that Drive shortcuts are better created using Advanced Drive Service / Drive API.

How it works:
Just `Logger.log()`—it's a "documentation snippet."

When to use:

- Repo documentation / reminders
- When teaching Drive automation patterns

Gotchas:

- Not actually creating a shortcut
- Requires enabling Advanced Drive service if you implement real shortcut creation

---

154) List recent files (last 7 days)

What it does:
Logs files updated within the last 7 days.

How it works:

- Sets a cutoff date
- Iterates search results and checks `getLastUpdated()`

When to use:

- Monitoring and audits
- Finding recent outputs

Gotchas:

- Searching broad Drive can be slow
- Better approach: include date constraints in query if possible

---

155) Export spreadsheet as XLSX

What it does:
Exports the active Google Sheet as an Excel `.xlsx` file and saves it to Drive.

How it works:

- Builds export URL
- Uses `ScriptApp.getOAuthToken()`
- `UrlFetchApp.fetch()` downloads file blob

- `DriveApp.createFile(blob)` saves it

When to use:

- Sending Excel files to clients
- Archiving snapshots

Gotchas:

- Requires UrlFetch authorization
- Export may fail if file is restricted by domain policies

---

156) Export doc as PDF

What it does:
Exports the active Google Doc as a PDF and saves it to Drive.

How it works:
Same pattern as 155 but using Docs export endpoint.

When to use:

- Publishing reports
- Generating printable deliverables

Gotchas:

- Requires UrlFetch + Docs permissions
- Export styling depends on doc formatting

---

157) Trash old files in a folder

What it does:
Trashes files older than N days in a specified folder.

How it works:
Compares `file.getLastUpdated()` to cutoff timestamp and calls `setTrashed(true)`.

When to use:

- Auto-cleanup for export folders
- Retention policies for logs/build artifacts

Gotchas:

- Destructive if misconfigured
- “Last updated” can change even if content changes slightly—choose date criteria carefully

---

158) Create “handoff” folder structure

What it does:
Creates a root folder and standard subfolders (Docs/Assets/Exports/Archive).

How it works:
`DriveApp.createFolder()` then loops array to create subfolders.

When to use:

- Client projects
- Repeatable internal workflows

Gotchas:

- Doesn’t apply sharing permissions automatically
- You may want to set editor/viewer permissions after creation

---

159) Save HTML file to Drive

What it does:
Creates a `.html` file in Drive with the given HTML.

How it works:
`DriveApp.createFile(name, content, MimeType.HTML)`.

When to use:

- Storing static report previews
- Building “viewable” output artifacts

Gotchas:

- By default it’s private; combine with sharing snippet (148) to publish
- HTML file may render differently depending on browser security rules

---

160) Clone folder structure only

What it does:
Copies an entire folder hierarchy (folders only) from one Drive folder to another.

How it works:

- Creates destination root folder
- Recursively loops subfolders and creates matching folders

When to use:

- Creating project templates
- Building consistent structures for teams

Gotchas:

- Does not copy files
- Deep recursion may hit time limits on huge folder trees

Docs + Slides (161–180)

161) Doc template placeholders

What it does:
Copies a template doc and replaces placeholder tokens (like `{{TITLE}}`) with real values.

How it works:

- `makeCopy()`
- Open doc with `DocumentApp.openById()`
- `body.replaceText()` per placeholder

When to use:

- Contract generation
- Reports, certificates, personalized documents

Gotchas:

- Placeholders must match exactly
- `replaceText()` uses regex—special characters in placeholder text can behave unexpectedly

---

162) Insert Sheet range as Doc table

What it does:
Creates a new doc and inserts a table populated from a sheet range.

How it works:
Reads `getValues()` then `doc.getBody().appendTable(values)`.

When to use:

- Turning spreadsheet summaries into printable docs
- Automated reporting

Gotchas:

- Large tables can be slow
- Formatting of the table may need extra styling

---

163) Add header text

What it does:
Adds a header to the doc with “DRAFT” style text.

How it works:
`doc.addHeader()` then `appendParagraph()`.

When to use:

- Watermark-like labels
- Draft/Confidential headings

Gotchas:

- Doesn’t place diagonal watermark; it’s a header only
- Header exists on every page depending on doc settings

---

164) Collapse multiple newlines

What it does:
Reduces excessive blank lines in a doc (3+ newlines → 2).

How it works:
Uses `editAsText().replaceText(pattern, replacement)`.

When to use:

- Cleanup after pasting content
- Normalizing AI-generated text

Gotchas:

- `replaceText` patterns are regex-like; test on a copy
- Can remove intended spacing if user wanted big gaps

---

165) Remove inline images

What it does:
Deletes inline images from a Google Doc.

How it works:
Loops through body children and removes elements where type is inline image.

When to use:

- Stripping images before exporting
- Creating “text-only” versions

Gotchas:

- Only catches certain image placements (depends on how images are embedded)

- Removing elements changes indices—looping backwards is safest

---

166) Extract headings to summary doc

What it does:
Creates a new doc that lists all headings from the active doc.

How it works:
Reads paragraphs, filters those where heading is HEADING*, and outputs their text.

When to use:

- Auto table-of-contents drafts
- Document outlines for review

Gotchas:

- Only headings set via Docs heading styles are detected
- Doesn't include nested heading levels unless they're styled

---

167) Create Slides deck (title + agenda)

What it does:
Creates a new Slides presentation with a title slide and an agenda slide.

How it works:
Uses `SlidesApp.create()` then inserts text boxes/placeholders.

When to use:

- Auto deck generation
- Standard meeting templates

Gotchas:

- Layout placeholders vary by theme
- You may need to adjust coordinates for different themes

---

168) Duplicate slide N times

What it does:
Duplicates a slide multiple times to quickly generate repeated template slides.

How it works:
Fetches slide 0 and appends copies in a loop.

When to use:

- Creating repeated pages for a batch presentation
- Generating “per client/per product” slides

Gotchas:

- Copies include all content exactly
- Can bloat deck quickly

---

169) Replace placeholder text in Slides

What it does:
Replaces token text (`{{NAME}}`) across the entire deck.

How it works:
`presentation.replaceAllText(find, replace)`.

When to use:

- Personalized decks

- Automating “mail merge” style slide creation

Gotchas:

- Only replaces text that matches exactly
- Doesn’t handle partial formatting edge cases if text runs are split

---

170) Add speaker notes

What it does:
Adds speaker notes to every slide.

How it works:
`slide.getNotesPage().getSpeakerNotesShape().getText().setText(...)`.

When to use:

- Scripted presentations
- Training decks

Gotchas:

- Notes overwrite any existing notes
- Some slides may not have expected notes shape if corrupted/unusual theme

---

171) Export Slides to PDF

What it does:
Exports a Slides presentation as a PDF and saves it to Drive.

How it works:
Authenticated `UrlFetchApp.fetch()` to Slides export endpoint; saves blob.

When to use:

- PDF handouts
- Sharing read-only versions

Gotchas:

- Requires UrlFetch scopes
- Large decks may be slow or time out

---

172) Doc with horizontal-rule sections

What it does:
Creates a doc with multiple sections separated by horizontal rules.

How it works:
Appends paragraphs + `appendHorizontalRule()` in a loop.

When to use:

- Structured reports
- Repeatable section templates

Gotchas:

- Doesn't style sections automatically (headings are simple unless set)

---

173) Doc → simple HTML

What it does:
Creates a basic HTML string from doc text, wrapping each line as `<p>`.

How it works:
`getBody().getText()` → split on `\n` → join paragraphs.

When to use:

- Simple export format for websites/emails
- Quick conversions

Gotchas:

- Loses rich formatting (bold, links, lists)
- Needs HTML escaping for special characters if used externally

---

174) Page break after Heading 2

What it does:
Inserts a page break after every Heading 2 paragraph.

How it works:
Scans paragraphs, finds Heading2, inserts page break after the element.

When to use:

- Chapter formatting
- Auto structuring long docs

Gotchas:

- Can create too many pages if headings are frequent
- Always test on a copy first

---

175) Meeting minutes template

What it does:
Creates a new doc with common meeting minutes sections.

How it works:
`DocumentApp.create()` then appends fixed headings/labels.

When to use:

- Standardizing meeting notes
- Teams that forget structure

Gotchas:

- Doesn't capture attendees automatically (could be extended with Calendar integration)

---

176) Selection to bullets

What it does:
Turns selected paragraphs into bullet list format.

How it works:
Gets current selection → iterates selected elements → sets parent paragraph bullet.

When to use:

- Quick formatting tool
- Cleaning messy notes

Gotchas:

- Requires an active selection (otherwise does nothing)
- Not all selected elements map neatly to paragraphs

---

177) Table with styled header

What it does:
Creates a table and styles the first row like a header (background + bold).

How it works:
Builds table, then formats each cell in row 0.

When to use:

- Printable reports
- Standardized doc tables

Gotchas:

- Color values must be valid hex
- More styling requires iterating paragraphs/text runs inside cells

---

178) Slides from list of titles

What it does:
Creates one slide per title in an array.

How it works:
Creates a deck, appends slides in a loop, removes default slide.

When to use:

- Outline-driven deck generation
- Curriculum/training deck skeletons

Gotchas:

- Title placeholder depends on layout/theme
- You'll likely want to add body content afterward

---

179) Export Doc to DOCX

What it does:
Exports Google Doc as Microsoft Word `.docx`.

How it works:
Authenticated fetch to `/export?format=docx`, saves blob.

When to use:

- Clients requiring Word format
- Cross-platform editing workflows

Gotchas:

- Requires UrlFetch scopes
- Some formatting may not translate perfectly to Word

---

180) Normalize spaces in Slides

What it does:
Cleans slide text by collapsing repeated whitespace to single spaces.

How it works:
Iterates slides → shapes → reads text → `.replace(/\s+/g,' ')`.

When to use:

- Fixing messy pasted content in slides
- Consistent formatting before export

Gotchas:

- Removes intentional spacing and line breaks if you're not careful (this keeps only spaces, not structure)

Gmail + Calendar + Web + Utilities (181–200)

181) Create nested Gmail label

What it does:
Creates (or retrieves) a hierarchical Gmail label like `Automation/Processed`.

How it works:
Checks for label existence, creates if missing.

When to use:

- Email processing workflows
- Categorizing automation results

Gotchas:

- Requires Gmail authorization
- Label naming conventions matter; Gmail treats `/` as hierarchy

---

182) Mark matching emails as read

What it does:
Finds inbox threads matching a Gmail query and marks them read.

How it works:
`GmailApp.search(query, start, max)` → `.markRead()`.

When to use:

- Auto-triage newsletters/alerts
- Clearing processed items

Gotchas:

- Query must be correct or you'll mark wrong mail as read
- Thread-based; affects entire conversation

---

183) Forward matching emails

What it does:
Finds threads, grabs the latest message, forwards it to another address.

How it works:
`thread.getMessages().pop().forward(email)`.

When to use:

- Escalation workflows
- Forwarding alerts to a team inbox

Gotchas:

- Replace recipient address
- Rate limits apply for sending/forwarding

---

184) Save email bodies to Doc

What it does:
Creates a doc digest of recent emails including subject and excerpt.

How it works:
Search threads → pick latest message → write subject + body excerpt into doc.

When to use:

- Daily review digest

- Archiving key emails into a doc

Gotchas:

- Uses plain body; formatting is lost
- Large emails truncated (by design)

---

185) Calendar event with reminders

What it does:
Creates an event and adds popup + email reminders.

How it works:
Creates event with start/end, then `addPopupReminder()` and `addEmailReminder()`.

When to use:

- Auto scheduling habits
- Booking “focus blocks”

Gotchas:

- Requires Calendar permissions
- Reminders depend on user calendar settings and delivery

---

186) Delete calendar events by search

What it does:
Deletes events matching a search phrase in a time window.

How it works:
`getEvents(start,end,{search})` → `deleteEvent()`.

When to use:

- Cleaning temporary/test events
- Removing old automation artifacts

Gotchas:

- Destructive (no easy undo)
- Make search phrase very specific to avoid deleting real events

---

187) Create hourly trigger

What it does:
Creates an Apps Script time-based trigger to run a function hourly.

How it works:
`ScriptApp.newTrigger(fn).timeBased().everyHours(1).create()`.

When to use:

- Scheduled automation tasks
- Regular sync jobs

Gotchas:

- Can create duplicates if run repeatedly (you might want a “trigger exists” check)
- Triggers run as the installing user

---

188) Delete triggers by handler

What it does:
Deletes all triggers that call a specific handler function.

How it works:
Iterates `getProjectTriggers()` and deletes matching ones.

When to use:

- Cleanup after testing
- Resetting trigger configurations

Gotchas:

- Removes all triggers for that handler (even ones you wanted)
- Doesn't differentiate trigger types unless you add checks

---

189) User preferences with UserProperties

What it does:
Stores and reads small per-user settings (ex: theme = dark).

How it works:
`PropertiesService.getUserProperties()` persists key/value for each user.

When to use:

- Personalizing UI behaviors
- Storing user-specific config without a database

Gotchas:

- Value sizes are limited
- Not shared across users (by design)

---

190) Rate limit per user (1/min)

What it does:
Prevents a user from running something too frequently.

How it works:
Stores last call timestamp in user properties and blocks if too soon.

When to use:

- Web apps / buttons to prevent spam
- Protecting quotas

Gotchas:

- Time-based only; doesn't consider workload duration
- Multiple users have independent limits

---

191) HMAC signature

What it does:
Generates an HMAC SHA-256 signature (common for webhook validation).

How it works:
`computeHmacSha256Signature(payload, secret)` returns bytes → convert to hex string.

When to use:

- Signing outgoing webhook payloads
- Verifying incoming requests (you'd compare signatures)

Gotchas:

- Keep secret out of code if possible (use PropertiesService)
- Must match the exact encoding the other system expects

---

192) “Encrypt/decrypt” demo note

What it does:
Demonstrates deriving a key and producing a keyed hash-like output (not true reversible encryption).

How it works:
Builds a derived key from SHA-256 and computes HMAC signature, base64 encodes it.

When to use:

- Teaching / illustrating hashing vs encryption
- Creating non-reversible integrity tokens

Gotchas:

- Not actual AES encryption and not decryptable
- For real encryption, you’d need a proper AES implementation pattern

---

193) JSON response helper

What it does:
Returns a JSON HTTP response from a web app.

How it works:

`ContentService.createTextOutput(JSON.stringify(data)).setMimeType(JSON)`.

When to use:

- Apps Script web app APIs
- Quick endpoints for dashboards/integrations

Gotchas:

- Doesn’t add CORS headers (needed for browser fetch from other origins)
- Doesn’t set HTTP status codes (ContentService is limited)

---

194) doGet router (web app)

What it does:
Creates a simple API router that responds differently based on `?route=` parameter.

How it works:

- Reads `e.parameter.route`
- If `ping`: returns server time
- If `echo`: returns all query params
- Else: returns error JSON

When to use:

- One endpoint with multiple actions
- Prototyping mini APIs

Gotchas:

- Must deploy as a Web App
- For sensitive routes, add authentication checks (token / session)

---

195) Fetch JSON with errors

What it does:
Calls a URL, checks response code, parses JSON, logs result.

How it works:

- `UrlFetchApp.fetch()`
- `getResponseCode()` validation
- `JSON.parse(getContentText())`

When to use:

- Integrations with external APIs
- Data ingestion scripts

Gotchas:

- Add timeouts/retries for production
- External endpoints can fail or rate-limit you

---

196) POST JSON

What it does:
Sends a JSON payload via HTTP POST and logs the response.

How it works:
`UrlFetchApp.fetch(url, {method:'post', contentType:'application/json', payload:JSON.stringify(obj)})`.

When to use:

- Sending data to webhook endpoints
- Triggering workflows (Zapier/Make/custom APIs)

Gotchas:

- Some APIs require Authorization headers
- Handle non-200 responses explicitly

---

197) Publish HTML + email link

What it does:
Creates an HTML report file, makes it viewable via link, emails the URL to the user.

How it works:

- `DriveApp.createFile()` HTML
- `setSharing()` public link
- `GmailApp.sendEmail()` send link

When to use:

- Automated “daily report” delivery
- Sharing read-only HTML dashboards

Gotchas:

- Public sharing may be blocked by domain policy
- Email sending quotas apply

---

198) Job queue using Script Properties

What it does:
Implements a basic queue system (enqueue & dequeue) stored as JSON in Script Properties.

How it works:

- Reads `QUEUE` property (JSON array)
- Pushes a job object
- Saves back to properties
- Dequeue removes first job and logs it

When to use:

- Batch processing when you can't do everything in one run
- Building lightweight background-like workflows

Gotchas:

- Properties size limits: queue can't grow forever
- Use LockService if multiple triggers can enqueue at the same time

---

199) Sheet logger

What it does:
Writes logs into a sheet named `Logs` (timestamp, user, message).

How it works:
Creates `Logs` sheet if missing, then `appendRow(...)`.

When to use:

- Debugging automations
- Auditing script behavior

Gotchas:

- AppendRow becomes slow at very large scale

- Consider batching logs (write multiple rows at once)

---

200) End-to-end pipeline: Sheet → Doc → PDF → Email

What it does:
Automates a full export workflow: reads sheet data, generates a doc report, exports as PDF, saves it, emails the PDF link.

How it works:

- Reads values from a range
- Creates doc and writes title + table
- Converts doc file blob to PDF
- Saves PDF to Drive
- Emails the Drive URL

When to use:

- Automated reporting for stakeholders
- “One click” report generation for recurring meetings

Gotchas:

- Requires multiple permissions (Sheets, Docs, Drive, Gmail)
- Large tables can cause timeouts; reduce range or paginate
- Email quotas apply; consider sending to a group alias instead of many recipients

```
/************************
  101–140: Sheets (advanced patterns)
************************/

// 101) Create a custom menu
function ex101_customMenu() {
  SpreadsheetApp.getUi()
    .createMenu('Automation')
    .addItem('Run Cleanup', 'ex112_cleanupTrim')
    .addToUi();
}

// 102) Add a sidebar UI
function ex102_showSidebar() {
  const html = HtmlService.createHtmlOutput('<h3>Sidebar</h3><p>Ready.</p>');
  SpreadsheetApp.getUi().showSidebar(html);
}

// 103) Find & replace across entire sheet
function ex103_findReplaceSheet() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.createTextFinder('foo').replaceAllWith('bar');
}
```

```
// 104) Freeze first column + header row
function ex104_freezeHeaderAndFirstCol() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.setFrozenRows(1);
  sh.setFrozenColumns(1);
}

// 105) Auto-size columns (A:Z)
function ex105_autoResize() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.autoResizeColumns(1, Math.min(26, sh.getLastColumn()));
}

// 106) Apply data validation dropdown list
function ex106_dropdownValidation() {
  const sh = SpreadsheetApp.getActiveSheet();
  const rule = SpreadsheetApp.newDataValidation()
    .requireValueInList(['New', 'In Progress', 'Done'], true)
    .setAllowInvalid(false)
    .build();
  sh.getRange('C2:C').setDataValidation(rule);
}

// 107) Add checkbox column
```

```
function ex107_addCheckboxes() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange('D2:D').insertCheckboxes();
}

// 108) Protect sheet except an input range
function ex108_protectExceptRange() {
  const sh = SpreadsheetApp.getActiveSheet();
  const p = sh.protect().setDescription('Protected except inputs');
  p.setUnprotectedRanges([sh.getRange('B2:D20')]);
}

// 109) Create a filter view-style filter (basic filter)
function ex109_createFilter() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getDataRange();
  if (!sh.getFilter()) r.createFilter();
}

// 110) Sort by multiple columns
function ex110_multiSort() {
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getDataRange();
  range.sort([{column: 2, ascending: true}, {column: 1, ascending: true}]);
```

```
}

// 111) Split text to columns by delimiter
function ex111_splitToColumns() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getRange('A2:A' + sh.getLastRow());
  const values = r.getValues().map(([v]) => String(v || '').split('|'));
  const maxLen = Math.max(...values.map(a => a.length), 1);
  const out = values.map(a => a.concat(Array(maxLen - a.length).fill('')));
  sh.getRange(2, 2, out.length, maxLen).setValues(out);
}

// 112) Cleanup: trim + collapse spaces in a range
function ex112_cleanupTrim() {
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getRange(2,1,Math.max(1, sh.getLastRow()-1), sh.getLastColumn());
  const data = range.getValues().map(row => row.map(v => {
    if (typeof v !== 'string') return v;
    return v.trim().replace(/\s+/g, ' ');
  }));
  range.setValues(data);
}

// 113) Convert all text to Title Case in column
```

```
function ex113_titleCaseColA() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getRange(2,1,Math.max(1, sh.getLastRow()-1),1);
  const out = r.getValues().map(([v]) => [String(v||'').toLowerCase().replace(/\b\w/g, c =>
c.toUpperCase())]);
  r.setValues(out);
}

// 114) Create a timestamp column when status changes (installable onEdit)
function ex114_onEditTimestamp(e) {
  const sh = e.range.getSheet();
  if (sh.getName() !== SpreadsheetApp.getActiveSheet().getName()) return;
  const statusCol = 3, tsCol = 4; // C status, D timestamp
  if (e.range.getColumn() === statusCol && e.range.getRow() > 1) {
    sh.getRange(e.range.getRow(), tsCol).setValue(new Date());
  }
}

// 115) Copy only visible rows (after filter) to another sheet
function ex115_copyVisibleRows() {
  const ss = SpreadsheetApp.getActive();
  const src = ss.getActiveSheet();
  const dst = ss.getSheetByName('Visible Copy') || ss.insertSheet('Visible Copy');
  const range = src.getDataRange();
  const values = range.getValues();
```

```
  const out = [];
  for (let i=0;i<values.length;i++){
    if (!src.isRowHiddenByFilter(i+1) && !src.isRowHiddenByUser(i+1)) out.push(values[i]);
  }
  dst.clear();
  dst.getRange(1,1,out.length,out[0].length).setValues(out);
}

// 116) Create pivot table (basic)
function ex116_createPivot() {
  const ss = SpreadsheetApp.getActive();
  const src = ss.getActiveSheet();
  const pivot = ss.getSheetByName('Pivot') || ss.insertSheet('Pivot');
  pivot.clear();
  const sourceRange = src.getDataRange();
  const anchor = pivot.getRange('A1');
  const pt = anchor.createPivotTable(sourceRange);
  pt.addRowGroup(1);
  pt.addPivotValue(2, SpreadsheetApp.PivotTableSummarizeFunction.COUNTA);
}

// 117) Add conditional formatting: top 10%
function ex117_cfTopTenPercent() {
  const sh = SpreadsheetApp.getActiveSheet();
```

```
  const rng = sh.getRange(2,2,Math.max(1, sh.getLastRow()-1),1);
  const rule = SpreadsheetApp.newConditionalFormatRule()
    .whenNumberGreaterThan(
      SpreadsheetApp.newConditionalFormatRule()
        .whenFormulaSatisfied('=TRUE') // dummy; we'll just use built-in top-10% style via formula
        .build()
    );
  // Easier: formula rule approximating top 10% cutoff
  const formula = '=B2>=PERCENTILE($B$2:$B,0.9)';
  const r2 = SpreadsheetApp.newConditionalFormatRule()
    .whenFormulaSatisfied(formula)
    .setBackground('#d1fae5')
    .setRanges([rng])
    .build();
  sh.setConditionalFormatRules([...sh.getConditionalFormatRules(), r2]);
}

// 118) Create a named range for current data table
function ex118_createNamedRange() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();
  const range = sh.getDataRange();
  ss.setNamedRange('DATA_TABLE', range);
}
```

```
// 119) Convert formulas to values in selection
function ex119_formulasToValues() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getActiveRange();
  r.copyTo(r, {contentsOnly: true});
}

// 120) Add hyperlinks in column B from URLs in column A
function ex120_makeHyperlinks() {
  const sh = SpreadsheetApp.getActiveSheet();
  const last = sh.getLastRow();
  const urls = sh.getRange(2,1,Math.max(1,last-1),1).getValues().flat();
  const out = urls.map(u => [u ? `=HYPERLINK("${u}","Open")` : '']);
  sh.getRange(2,2,out.length,1).setValues(out);
}

// 121) Generate random IDs
function ex121_generateIds() {
  const sh = SpreadsheetApp.getActiveSheet();
  const last = sh.getLastRow();
  const out = Array.from({length: Math.max(1,last-1)}, () => [Utilities.getUuid()]);
  sh.getRange(2,5,out.length,1).setValues(out);
}
```

```
// 122) Create a “missing values” report
function ex122_missingValuesReport() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();
  const rep = ss.getSheetByName('Missing Report') || ss.insertSheet('Missing Report');
  const data = sh.getDataRange().getValues();
  const header = data[0] || [];
  const misses = [];
  for (let r=1;r<data.length;r++){
    for (let c=0;c<header.length;c++){
      if (data[r][c] === '' || data[r][c] == null) misses.push([r+1, c+1, header[c] || '', 'blank']);
    }
  }
  rep.clear();
  rep.getRange(1,1,1,4).setValues([['Row','Col','Header','Issue']]);
  if (misses.length) rep.getRange(2,1,misses.length,4).setValues(misses);
}

// 123) Sync two sheets by ID (left join)
function ex123_syncById() {
  const ss = SpreadsheetApp.getActive();
  const a = ss.getSheetByName('SheetA') || ss.getActiveSheet();
  const b = ss.getSheetByName('SheetB');
```

```
  if (!b) throw new Error('Create SheetB with IDs in col A');
  const aData = a.getDataRange().getValues();
  const bData = b.getDataRange().getValues();
  const bMap = new Map(bData.slice(1).map(r => [String(r[0]).trim(), r]));
  const out = [aData[0].concat(bData[0].slice(1))];
  aData.slice(1).forEach(r => {
    const key = String(r[0]).trim();
    const match = bMap.get(key);
    out.push(r.concat(match ? match.slice(1) : Array(bData[0].length-1).fill('')));
  });
  const dst = ss.getSheetByName('Synced') || ss.insertSheet('Synced');
  dst.clear();
  dst.getRange(1,1,out.length,out[0].length).setValues(out);
}

// 124) Write a chart to sheet (basic column chart)
function ex124_createChart() {
  const sh = SpreadsheetApp.getActiveSheet();
  const range = sh.getRange('A1:B10');
  const chart = sh.newChart()
    .setChartType(Charts.ChartType.COLUMN)
    .addRange(range)
    .setPosition(1, 4, 0, 0)
    .build();
```

```
  sh.insertChart(chart);
}

// 125) Remove all charts from sheet
function ex125_removeCharts() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getCharts().forEach(c => sh.removeChart(c));
}

// 126) Duplicate sheet and rename with timestamp
function ex126_duplicateSheet() {
  const ss = SpreadsheetApp.getActive();
  const src = ss.getActiveSheet();
  const copy = src.copyTo(ss);
  copy.setName(src.getName() + ' - ' + Utilities.formatDate(new Date(),
ss.getSpreadsheetTimeZone(), 'yyyyMMdd-HHmm'));
}

// 127) Create a “daily log” row in a sheet
function ex127_appendDailyLog() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.appendRow([new Date(), Session.getActiveUser().getEmail(), 'Daily log entry']);
}

// 128) Lock a range for 10 minutes (soft lock via cache)
```

```
function ex128_softLock() {
  const cache = CacheService.getScriptCache();
  const key = 'LOCK:PROCESS';
  if (cache.get(key)) throw new Error('Locked: try again later.');
  cache.put(key, '1', 600);
  // ... do work ...
}

// 129) Generate a “run report” in a new sheet
function ex129_runReport() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.insertSheet('Run Report ' + Date.now());
  sh.getRange('A1').setValue('Run at');
  sh.getRange('B1').setValue(new Date());
}

// 130) Convert currency strings to numbers (e.g., "$1,234.56")
function ex130_currencyToNumber() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getRange('B2:B' + sh.getLastRow());
  const out = r.getValues().map(([v]) => {
    const n = Number(String(v||'').replace(/[^0-9.-]/g,''));
    return [isNaN(n) ? '' : n];
  });
```

```
  r.setValues(out);
}

// 131) Remove duplicates by column (keep first)
function ex131_removeDuplicatesByColA() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getDataRange().removeDuplicates([1]);
}

// 132) Add alternating row colors
function ex132_alternateColors() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange(1,1,sh.getLastRow(),sh.getLastColumn()).applyRowBanding();
}

// 133) Create a “top N” extraction to another sheet
function ex133_topN() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();
  const data = sh.getDataRange().getValues();
  const header = data.shift();
  const sorted = data.sort((a,b)=> (b[1]||0)-(a[1]||0)).slice(0,10);
  const out = [header, ...sorted];
  const dst = ss.getSheetByName('Top10') || ss.insertSheet('Top10');
```

```
  dst.clear(); dst.getRange(1,1,out.length,out[0].length).setValues(out);
}

// 134) Detect outliers (z-score) in column B
function ex134_detectOutliers() {
  const sh = SpreadsheetApp.getActiveSheet();
  const vals = sh.getRange('B2:B' +
sh.getLastRow()).getValues().flat().map(Number).filter(v=>!isNaN(v));
  const mean = vals.reduce((a,b)=>a+b,0)/Math.max(vals.length,1);
  const sd = Math.sqrt(vals.reduce((s,v)=>s+Math.pow(v-mean,2),0)/Math.max(vals.length,1));
  const r = sh.getRange('B2:B' + sh.getLastRow());
  const out = r.getValues().map(([v])=>{
    const n=Number(v); if(isNaN(n)||sd===0) return [''];
    const z=(n-mean)/sd;
    return [Math.abs(z)>=3 ? 'OUTLIER' : ''];
  });
  sh.getRange(2,3,out.length,1).setValues(out); // write to col C
}

// 135) Convert date text to Date objects in column A
function ex135_parseDates() {
  const sh = SpreadsheetApp.getActiveSheet();
  const r = sh.getRange('A2:A' + sh.getLastRow());
  const out = r.getValues().map(([v]) => {
    const d = new Date(v);
```

```
    return [isNaN(d.getTime()) ? '' : d];
  });
  r.setValues(out);
}

// 136) Create a monthly summary sheet (counts by month in column A dates)
function ex136_monthlySummary() {
  const ss = SpreadsheetApp.getActive();
  const sh = ss.getActiveSheet();
  const dates = sh.getRange('A2:A' + sh.getLastRow()).getValues().flat().map(v=>new
Date(v)).filter(d=>!isNaN(d));
  const tz = ss.getSpreadsheetTimeZone();
  const map = {};
  dates.forEach(d=>{
    const k = Utilities.formatDate(d, tz, 'yyyy-MM');
    map[k] = (map[k]||0)+1;
  });
  const out = [['Month','Count'], ...Object.entries(map).sort().map(([k,v])=>[k,v])];
  const dst = ss.getSheetByName('Monthly Summary') || ss.insertSheet('Monthly Summary');
  dst.clear(); dst.getRange(1,1,out.length,2).setValues(out);
}

// 137) Import a CSV string into the sheet
function ex137_importCsvString() {
  const csv = "Name,Score\nAva,95\nNoah,88";
```

```
  const rows = Utilities.parseCsv(csv);
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getRange(1,1,rows.length,rows[0].length).setValues(rows);
}

// 138) Export entire sheet to JSON (array of objects)
function ex138_exportToJson() {
  const sh = SpreadsheetApp.getActiveSheet();
  const data = sh.getDataRange().getValues();
  const headers = data.shift().map(String);
  const objs = data.map(r => Object.fromEntries(headers.map((h,i)=>[h, r[i]])));
  Logger.log(JSON.stringify(objs, null, 2));
}

// 139) Add comments (notes) from column E onto column A
function ex139_notesFromColE() {
  const sh = SpreadsheetApp.getActiveSheet();
  const last = sh.getLastRow();
  const notes = sh.getRange(2,5,Math.max(1,last-1),1).getValues().flat();
  const target = sh.getRange(2,1,notes.length,1);
  target.setNotes(notes.map(n=>[String(n||'')]));
}

// 140) Clear content but keep formatting
```

```
function ex140_clearContentKeepFormatting() {
  const sh = SpreadsheetApp.getActiveSheet();
  sh.getDataRange().clearContent();
}

/************************
  141–160: Drive (advanced file workflows)
************************/

// 141) Create a folder and return URL
function ex141_createFolder() {
  const f = DriveApp.createFolder('New Folder ' + Date.now());
  Logger.log(f.getUrl());
}

// 142) List files in a folder (names)
function ex142_listFilesInFolder() {
  const folderId = 'PASTE_FOLDER_ID';
  const folder = DriveApp.getFolderById(folderId);
  const it = folder.getFiles();
  while (it.hasNext()) Logger.log(it.next().getName());
}

// 143) Copy a file to another folder
```

```
function ex143_copyFileToFolder() {
  const fileId = 'PASTE_FILE_ID';
  const folderId = 'PASTE_FOLDER_ID';
  const copy = DriveApp.getFileById(fileId).makeCopy(DriveApp.getFolderById(folderId));
  Logger.log(copy.getUrl());
}

// 144) Move a file to another folder
function ex144_moveFileToFolder() {
  const fileId = 'PASTE_FILE_ID';
  const folderId = 'PASTE_FOLDER_ID';
  const file = DriveApp.getFileById(fileId);
  const target = DriveApp.getFolderById(folderId);
  const parents = file.getParents();
  target.addFile(file);
  while (parents.hasNext()) parents.next().removeFile(file);
}

// 145) Find files by partial name
function ex145_searchByName() {
  const q = 'name contains "Report" and trashed=false';
  const it = DriveApp.searchFiles(q);
  let n=0;
  while (it.hasNext() && n<20) { const f=it.next(); Logger.log(f.getName() + ' ' + f.getUrl()); n++; }
```

```
}

// 146) Create a text file in a folder
function ex146_createTextFile() {
  const folderId='PASTE_FOLDER_ID';
  const folder=DriveApp.getFolderById(folderId);
  const file=folder.createFile('notes.txt','Hello world','text/plain');
  Logger.log(file.getUrl());
}

// 147) Update a text file’s contents
function ex147_overwriteTextFile() {
  const fileId='PASTE_FILE_ID';
  const file=DriveApp.getFileById(fileId);
  file.setContent('Updated at ' + new Date());
}

// 148) Make a file “anyone with link can view”
function ex148_makePublicView() {
  const fileId='PASTE_FILE_ID';
  DriveApp.getFileById(fileId).setSharing(DriveApp.Access.ANYONE_WITH_LINK,
DriveApp.Permission.VIEW);
}

// 149) Create a Drive “manifest” (name → id) as JSON file
```

```
function ex149_createManifestJson() {
  const folderId='PASTE_FOLDER_ID';
  const folder=DriveApp.getFolderById(folderId);
  const it=folder.getFiles();
  const manifest = {};
  while(it.hasNext()){
    const f=it.next();
    manifest[f.getName()] = f.getId();
  }
  const blob = Utilities.newBlob(JSON.stringify(manifest, null, 2), 'application/json',
'manifest.json');
  const file = folder.createFile(blob);
  Logger.log(file.getUrl());
}


// 150) Zip files in a folder and save the zip
function ex150_zipFolderFiles() {
  const folderId='PASTE_FOLDER_ID';
  const folder=DriveApp.getFolderById(folderId);
  const it=folder.getFiles();
  const blobs=[];
  while(it.hasNext()){
    const f=it.next();
    blobs.push(f.getBlob().setName(f.getName()));
  }
```

```
  const zipBlob = Utilities.zip(blobs, 'bundle.zip');
  const zipFile = folder.createFile(zipBlob);
  Logger.log(zipFile.getUrl());
}

// 151) Delete empty folders (one level deep)
function ex151_deleteEmptySubfolders() {
  const rootId='PASTE_FOLDER_ID';
  const root=DriveApp.getFolderById(rootId);
  const subs=root.getFolders();
  while(subs.hasNext()){
    const f=subs.next();
    if(!f.getFiles().hasNext() && !f.getFolders().hasNext()){
      f.setTrashed(true);
    }
  }
}

// 152) Rename all files by replacing substring
function ex152_renameReplace() {
  const folderId='PASTE_FOLDER_ID';
  const folder=DriveApp.getFolderById(folderId);
  const it=folder.getFiles();
  while(it.hasNext()){
```

```
    const f=it.next();
    f.setName(f.getName().replace('Draft','Final'));
  }
}

// 153) Create a shortcut file (requires Drive API in many cases; note)
function ex153_shortcutNote() {
  Logger.log('Drive shortcuts are best created via Drive API (Advanced Drive Service) with mimeType "application/vnd.google-apps.shortcut".');
}

// 154) List recent files (last 7 days) across Drive
function ex154_recentFiles() {
  const cutoff = new Date(Date.now() - 7*24*3600*1000);
  const it = DriveApp.searchFiles('trashed=false');
  let n=0;
  while(it.hasNext() && n<50){
    const f=it.next();
    if (f.getLastUpdated() >= cutoff) Logger.log(f.getLastUpdated() + ' ' + f.getName());
    n++;
  }
}

// 155) Copy a spreadsheet as XLSX to Drive
function ex155_exportSheetAsXlsx() {
```

```
  const ssId = SpreadsheetApp.getActive().getId();
  const url = 'https://docs.google.com/spreadsheets/d/' + ssId + '/export?format=xlsx';
  const token = ScriptApp.getOAuthToken();
  const resp = UrlFetchApp.fetch(url, {headers:{Authorization:'Bearer ' + token}});
  const file = DriveApp.createFile(resp.getBlob().setName('export.xlsx'));
  Logger.log(file.getUrl());
}

// 156) Copy a doc as PDF to Drive
function ex156_exportDocAsPdf() {
  const docId = DocumentApp.getActiveDocument().getId();
  const url = 'https://docs.google.com/document/d/' + docId + '/export?format=pdf';
  const token = ScriptApp.getOAuthToken();
  const resp = UrlFetchApp.fetch(url, {headers:{Authorization:'Bearer ' + token}});
  const file = DriveApp.createFile(resp.getBlob().setName('export.pdf'));
  Logger.log(file.getUrl());
}

// 157) Find and trash files older than N days in a folder
function ex157_trashOldInFolder() {
  const folderId='PASTE_FOLDER_ID', days=90;
  const cutoff=Date.now() - days*24*3600*1000;
  const folder=DriveApp.getFolderById(folderId);
  const it=folder.getFiles();
```

```
  while(it.hasNext()){
    const f=it.next();
    if(f.getLastUpdated().getTime() < cutoff) f.setTrashed(true);
  }
}

// 158) Create a shared “handoff” folder structure
function ex158_createHandoffStructure() {
  const root = DriveApp.createFolder('Handoff ' + Date.now());
  ['01-Docs','02-Assets','03-Exports','04-Archive'].forEach(n => root.createFolder(n));
  Logger.log(root.getUrl());
}

// 159) Save an HTML file to Drive
function ex159_saveHtmlFile() {
  const html = '<!doctype html><html><body><h1>Hello</h1></body></html>';
  const file = DriveApp.createFile('page.html', html, MimeType.HTML);
  Logger.log(file.getUrl());
}

// 160) Duplicate a Drive folder structure (folders only)
function ex160_cloneFolderStructureOnly() {
  const sourceId='PASTE_SOURCE_FOLDER_ID';
  const targetParentId='PASTE_TARGET_PARENT_FOLDER_ID';
```

```
  const src=DriveApp.getFolderById(sourceId);
  const parent=DriveApp.getFolderById(targetParentId);
  const dst=parent.createFolder(src.getName() + ' (Structure)');
  cloneFoldersOnly_(src, dst);
}
function cloneFoldersOnly_(src, dst){
  const subs=src.getFolders();
  while(subs.hasNext()){
    const sub=subs.next();
    const newSub=dst.createFolder(sub.getName());
    cloneFoldersOnly_(sub, newSub);
  }
}

/**********************
  161–180: Docs + Slides (document generation)
**********************/

// 161) Create a Doc from template placeholders
function ex161_docTemplateFromMap() {
  const templateId='PASTE_TEMPLATE_DOC_ID';
  const copy=DriveApp.getFileById(templateId).makeCopy('Generated Doc ' + Date.now());
  const doc=DocumentApp.openById(copy.getId());
  const body=doc.getBody();
```

```
  const map={'{{TITLE}}':'Automation Report','{{AUTHOR}}':'Lars','{{DATE}}':new
Date().toDateString()};

  Object.entries(map).forEach(([k,v])=>body.replaceText(k, String(v)));

  doc.saveAndClose();

  Logger.log(doc.getUrl());

}


// 162) Create a doc with a table of data from a sheet range

function ex162_docTableFromSheet() {

  const sh=SpreadsheetApp.getActiveSheet();

  const values=sh.getRange('A1:D10').getValues();

  const doc=DocumentApp.create('Table Doc ' + Date.now());

  doc.getBody().appendTable(values);

  Logger.log(doc.getUrl());

}


// 163) Add a watermark-like header text

function ex163_addHeaderText() {

  const doc=DocumentApp.getActiveDocument();

  const header=doc.addHeader();

  header.appendParagraph('DRAFT').setBold(true);

}


// 164) Replace multiple newlines with single newline

function ex164_collapseNewlines() {
```

```
  const doc=DocumentApp.getActiveDocument();
  const t=doc.getBody().editAsText();
  t.replaceText('\\n{3,}', '\n\n');
}

// 165) Remove all images from a doc
function ex165_removeImages() {
  const body=DocumentApp.getActiveDocument().getBody();
  const n=body.getNumChildren();
  for(let i=n-1;i>=0;i--){
    const el=body.getChild(i);
    if (el.getType() === DocumentApp.ElementType.INLINE_IMAGE) el.removeFromParent();
  }
}

// 166) Extract headings into a summary doc
function ex166_headingsSummary() {
  const src=DocumentApp.getActiveDocument();
  const headings=src.getBody().getParagraphs()
    .filter(p=>String(p.getHeading()).includes('HEADING'))
    .map(p=>p.getText());
  const doc=DocumentApp.create('Headings Summary ' + Date.now());
  headings.forEach(h=>doc.getBody().appendParagraph(h));
  Logger.log(doc.getUrl());
```

```
}

// 167) Create a Slides deck with title slide + agenda
function ex167_createSlidesAgenda() {
  const pres=SlidesApp.create('Agenda Deck ' + Date.now());
  const title=pres.getSlides()[0];
  title.insertShape(SlidesApp.ShapeType.TEXT_BOX, 40, 60, 640, 80).getText().setText('Project Update');
  const agenda=pres.appendSlide(SlidesApp.PredefinedLayout.TITLE_AND_BODY);
agenda.getPlaceholder(SlidesApp.PlaceholderType.TITLE).asShape().getText().setText('Agenda');
  agenda.getPlaceholder(SlidesApp.PlaceholderType.BODY).asShape().getText().setText('• Status\n• Risks\n• Next steps');
  Logger.log(pres.getUrl());
}

// 168) Duplicate a slide N times
function ex168_duplicateSlideNTimes() {
  const pres=SlidesApp.getActivePresentation();
  const slide=pres.getSlides()[0];
  for(let i=0;i<5;i++) pres.appendSlide(slide);
}

// 169) Replace text across all slides
function ex169_replaceTextInSlides() {
```

```
  const pres=SlidesApp.getActivePresentation();
  pres.replaceAllText('{{NAME}}', 'Lars');
}

// 170) Add speaker notes to all slides
function ex170_addSpeakerNotes() {
  const pres=SlidesApp.getActivePresentation();
  pres.getSlides().forEach((s,i)=>{
    s.getNotesPage().getSpeakerNotesShape().getText().setText('Notes for slide ' + (i+1));
  });
}

// 171) Export active Slides to PDF and save to Drive
function ex171_exportSlidesPdf() {
  const presId=SlidesApp.getActivePresentation().getId();
  const url='https://docs.google.com/presentation/d/' + presId + '/export/pdf';
  const token=ScriptApp.getOAuthToken();
  const resp=UrlFetchApp.fetch(url,{headers:{Authorization:'Bearer ' + token}});
  const file=DriveApp.createFile(resp.getBlob().setName('deck.pdf'));
  Logger.log(file.getUrl());
}

// 172) Create a doc and add a horizontal rule between sections
function ex172_docSectionsWithRules() {
```

```
  const doc=DocumentApp.create('Sectioned Doc ' + Date.now());
  const body=doc.getBody();
  ['One','Two','Three'].forEach((name)=>{
    body.appendParagraph(name).setHeading(DocumentApp.ParagraphHeading.HEADING2);
    body.appendParagraph('Content...');
    body.appendHorizontalRule();
  });
  Logger.log(doc.getUrl());
}

// 173) Convert a Google Doc to HTML string (simple)
function ex173_docToHtmlSimple() {
  const doc=DocumentApp.getActiveDocument();
  const html = doc.getBody().getText().split('\n').map(p=>`<p>${p}</p>`).join('');
  Logger.log(html);
}

// 174) Insert a page break after each Heading 2
function ex174_pageBreakAfterH2() {
  const doc=DocumentApp.getActiveDocument();
  const body=doc.getBody();
  const pars=body.getParagraphs();
  for(let i=pars.length-1;i>=0;i--){
    if(pars[i].getHeading()===DocumentApp.ParagraphHeading.HEADING2){
```

```
      body.insertPageBreak(body.getChildIndex(pars[i])+1);
    }
  }
}

// 175) Create a “meeting minutes” doc template
function ex175_meetingMinutesDoc() {
  const doc=DocumentApp.create('Meeting Minutes ' + new Date().toDateString());
  const b=doc.getBody();
  b.appendParagraph('Meeting
Minutes').setHeading(DocumentApp.ParagraphHeading.HEADING1);
  b.appendParagraph('Attendees:');
  b.appendParagraph('Agenda:');
  b.appendParagraph('Decisions:');
  b.appendParagraph('Action Items:');
  Logger.log(doc.getUrl());
}

// 176) Convert selected text to bullet list
function ex176_selectionToBullets() {
  const doc=DocumentApp.getActiveDocument();
  const sel=doc.getSelection();
  if(!sel) return;
  sel.getRangeElements().forEach(re=>{
    const el=re.getElement();
```

```
    if(el.editAsText) {
      const p = el.getParent();
      if (p && p.asParagraph) p.asParagraph().setBullet(true);
    }
  });
}

// 177) Add a table with styled header row
function ex177_docTableWithHeaderStyle() {
  const doc=DocumentApp.create('Styled Table ' + Date.now());
  const body=doc.getBody();
  const table=body.appendTable([['Name','Score'],['Ava','95'],['Noah','88']]);
  const header=table.getRow(0);
  for(let i=0;i<header.getNumCells();i++){
    header.getCell(i).setBackgroundColor('#e5e7eb');
    header.getCell(i).editAsText().setBold(true);
  }
  Logger.log(doc.getUrl());
}

// 178) Create a Slides deck from a list of titles
function ex178_slidesFromList() {
  const titles=['Intro','Problem','Solution','Demo','Next Steps'];
  const pres=SlidesApp.create('Outline Deck ' + Date.now());
```

```
  titles.forEach(t=>{
    const s=pres.appendSlide(SlidesApp.PredefinedLayout.TITLE_ONLY);
    s.getPlaceholder(SlidesApp.PlaceholderType.TITLE).asShape().getText().setText(t);
  });
  pres.getSlides()[0].remove();
  Logger.log(pres.getUrl());
}

// 179) Export doc as DOCX to Drive
function ex179_exportDocAsDocx() {
  const docId=DocumentApp.getActiveDocument().getId();
  const url='https://docs.google.com/document/d/' + docId + '/export?format=docx';
  const token=ScriptApp.getOAuthToken();
  const resp=UrlFetchApp.fetch(url,{headers:{Authorization:'Bearer ' + token}});
  const file=DriveApp.createFile(resp.getBlob().setName('export.docx'));
  Logger.log(file.getUrl());
}

// 180) Replace multiple spaces in Slides text shapes
function ex180_slidesNormalizeSpaces() {
  const pres=SlidesApp.getActivePresentation();
  pres.getSlides().forEach(slide=>{
    slide.getPageElements().forEach(el=>{
      if(el.getPageElementType()===SlidesApp.PageElementType.SHAPE){
```

```
        const t=el.asShape().getText();
        const s=t.asString();
        t.setText(s.replace(/\s+/g,' ').trim());
      }
    });
  });
}

/************************
  181–200: Gmail, Calendar, Web, Utilities
************************/

// 181) Create a Gmail label hierarchy (Label/Sub)
function ex181_createNestedLabel() {
  const name='Automation/Processed';
  const label=GmailApp.getUserLabelByName(name) || GmailApp.createLabel(name);
  Logger.log(label.getName());
}

// 182) Mark matching emails as read
function ex182_markRead() {
  const threads=GmailApp.search('in:inbox subject:"Weekly Report"',0,50);
  threads.forEach(t=>t.markRead());
}
```

```
// 183) Forward matching emails to another address
function ex183_forwardEmails() {
  const threads=GmailApp.search('subject:"Action Required" newer_than:7d',0,10);
  threads.forEach(t=>t.getMessages().pop().forward('someone@example.com'));
}

// 184) Save email bodies to a Google Doc
function ex184_emailBodiesToDoc() {
  const threads=GmailApp.search('newer_than:3d',0,5);
  const doc=DocumentApp.create('Email Bodies ' + Date.now());
  const body=doc.getBody();
  threads.forEach(t=>{
    const m=t.getMessages().pop();
body.appendParagraph(m.getSubject()).setHeading(DocumentApp.ParagraphHeading.HEADING2);
    body.appendParagraph(m.getPlainBody().slice(0,2000));
    body.appendHorizontalRule();
  });
  Logger.log(doc.getUrl());
}

// 185) Create a calendar event with custom reminders
function ex185_eventWithReminders() {
```

```
  const cal=CalendarApp.getDefaultCalendar();
  const start=new Date(Date.now()+2*3600*1000);
  const end=new Date(start.getTime()+45*60000);
  const ev=cal.createEvent('Deep Work', start, end);
  ev.addPopupReminder(10);
  ev.addEmailReminder(30);
}

// 186) Delete calendar events by search (careful)
function ex186_deleteEventsBySearch() {
  const cal=CalendarApp.getDefaultCalendar();
  const start=new Date(Date.now()-7*24*3600*1000);
  const end=new Date(Date.now()+7*24*3600*1000);
  const events=cal.getEvents(start,end,{search:'Temp Event'});
  events.forEach(e=>e.deleteEvent());
}

// 187) Create a time-based trigger (every hour)
function ex187_createHourlyTrigger() {
  ScriptApp.newTrigger('ex127_appendDailyLog')
    .timeBased()
    .everyHours(1)
    .create();
}
```

```
// 188) Delete triggers by handler name
function ex188_deleteTriggersByHandler() {
  const handler='ex127_appendDailyLog';
  ScriptApp.getProjectTriggers().forEach(t=>{
    if(t.getHandlerFunction()===handler) ScriptApp.deleteTrigger(t);
  });
}

// 189) Store and retrieve user preferences (UserProperties)
function ex189_userPrefs() {
  const props=PropertiesService.getUserProperties();
  props.setProperty('theme','dark');
  Logger.log(props.getProperty('theme'));
}

// 190) Simple rate limiter (per user per minute)
function ex190_rateLimitPerMinute() {
  const props=PropertiesService.getUserProperties();
  const key='lastCall';
  const last=Number(props.getProperty(key) || 0);
  const now=Date.now();
  if(now-last < 60*1000) throw new Error('Rate limited. Try again in a minute.');
  props.setProperty(key, String(now));
```

```
}

// 191) Generate an HMAC signature (Webhook signing)
function ex191_hmacSignature() {
  const secret='CHANGE_ME';
  const payload='hello';
  const sig = Utilities.computeHmacSha256Signature(payload, secret);
  const hex = sig.map(b=>('0'+(b & 0xff).toString(16)).slice(-2)).join('');
  Logger.log(hex);
}

// 192) AES encrypt/decrypt a string (simple)
function ex192_encryptDecrypt() {
  const key = Utilities.base64EncodeWebSafe(Utilities.computeDigest(Utilities.DigestAlgorithm.SHA_256, 'passphrase'));
  const plaintext='secret message';
  const enc = Utilities.base64EncodeWebSafe(Utilities.computeHmacSha256Signature(plaintext, key)); // not true AES, but safe keyed hash demo
  Logger.log({plaintext, enc});
}

// 193) Web app JSON response helper
function ex193_json(data) {
  return ContentService.createTextOutput(JSON.stringify(data))
```

```
    .setMimeType(ContentService.MimeType.JSON);
}

// 194) Web app doGet router example
function doGet(e) {
  const route = (e.parameter.route || 'ping').toLowerCase();
  if (route === 'ping') return ex193_json({ok:true, route, time:new Date().toISOString()});
  if (route === 'echo') return ex193_json({ok:true, query:e.parameter});
  return ex193_json({ok:false, error:'unknown route'});
}

// 195) Fetch JSON with timeout + error handling
function ex195_fetchJson() {
  const url='https://httpbin.org/json';
  const resp=UrlFetchApp.fetch(url, {muteHttpExceptions:true, followRedirects:true});
  if(resp.getResponseCode()>=400) throw new Error(resp.getContentText());
  const json=JSON.parse(resp.getContentText());
  Logger.log(json);
}

// 196) Post JSON to an endpoint
function ex196_postJson() {
  const url='https://httpbin.org/post';
  const payload={event:'test', at:new Date().toISOString()};
```

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

// 197) Create an HTML file, publish link, and email it
function ex197_publishHtmlAndEmail() {
  const html='<h1>Report</h1><p>Generated ' + new Date() + '</p>';
  const file=DriveApp.createFile('report.html', html, MimeType.HTML);
  file.setSharing(DriveApp.Access.ANYONE_WITH_LINK, DriveApp.Permission.VIEW);
  GmailApp.sendEmail(Session.getActiveUser().getEmail(), 'HTML Report Link', file.getUrl());
}

// 198) Build a simple “job queue” using Script Properties
function ex198_enqueueJob() {
  const props=PropertiesService.getScriptProperties();
  const q=JSON.parse(props.getProperty('QUEUE') || '[]');
  q.push({id:Utilities.getUuid(), created:Date.now(), task:'sync'});
  props.setProperty('QUEUE', JSON.stringify(q));
}
```

```
function ex198_dequeueJob() {
  const props=PropertiesService.getScriptProperties();
  const q=JSON.parse(props.getProperty('QUEUE') || '[]');
  const job=q.shift();
  props.setProperty('QUEUE', JSON.stringify(q));
  Logger.log(job || 'no jobs');
}

// 199) Minimal logger that writes to a “Logs” sheet
function ex199_sheetLogger(message) {
  const ss=SpreadsheetApp.getActive();
  const sh=ss.getSheetByName('Logs') || ss.insertSheet('Logs');
  sh.appendRow([new Date(), Session.getActiveUser().getEmail(), message]);
}

// 200) End-to-end: read sheet → create doc → save PDF → email link
function ex200_sheetDocPdfEmail() {
  const ss=SpreadsheetApp.getActive();
  const sh=ss.getActiveSheet();
  const values=sh.getRange('A1:D10').getValues();
  const doc=DocumentApp.create('Export ' + Date.now());

doc.getBody().appendParagraph('Export').setHeading(DocumentApp.ParagraphHeading.HEADI
NG1);
  doc.getBody().appendTable(values);
```

```
  doc.saveAndClose();


  const docFile=DriveApp.getFileById(doc.getId());

  const pdf=docFile.getBlob().getAs(MimeType.PDF).setName(docFile.getName()+'.pdf');

  const pdfFile=DriveApp.createFile(pdf);

  GmailApp.sendEmail(Session.getActiveUser().getEmail(), 'Your PDF Export', 'Here is your
PDF: ' + pdfFile.getUrl());

}
```