Released under MIT License

1. Upload Excel file to Google docs
2. Convert to Google docs
3. Prepare the worksheet environment Tabs (S1 The Raw, S2 The Formatted, S3 Additional)
4. Copy the Google sheet in the worksheet environment S1
5. APPS Script NLP script to bring the needed Fields (Columns from S1 to S2)
6. APPS Script NLP formatting steps 1, 2 , 3, 4, etc....Small incremental changes
test then move on to next.

The NLP:

1.
Capitalize the first letter of every word in row 1 in the first sheet named S1
while maintaining the column order.

```
/**
* Capitalizes the first letter of every word in row 1 on the specified sheet
* while maintaining the column order.
*
* @param {string} sheetName - The name of the sheet to modify (default: "Sheet1")
*/
function capitalizeFirstLettersInRow1(sheetName) {
  // If no sheet name is provided, use Sheet1 as default
  sheetName = sheetName || "Sheet1";

  // Get the active spreadsheet
  var ss = SpreadsheetApp.getActiveSpreadsheet();

  // Get the specified sheet
  var sheet = ss.getSheetByName(sheetName);

  if (!sheet) {
    Logger.log("Sheet '" + sheetName + "' not found.");
    return;
  }
```

```
// Get the number of columns in the sheet
var lastColumn = sheet.getLastColumn();

if (lastColumn <= 0) {
  Logger.log("Sheet is empty. No data to modify.");
  return;
}

// Get row 1 data
var row1Data = sheet.getRange(1, 1, 1, lastColumn).getValues()[0];
Logger.log("Original row 1 data: " + JSON.stringify(row1Data));

// Process each cell in row 1
var modified = false;

for (var i = 0; i < row1Data.length; i++) {
  // Convert to string if it's not already
  var cellValue = String(row1Data[i]);

  if (cellValue && cellValue.trim() !== '') {
    var originalValue = cellValue;

    // More robust word splitting - handles multiple spaces, tabs, etc.
    var words = cellValue.split(/\s+/);

    // Capitalize the first letter of each word
    for (var j = 0; j < words.length; j++) {
      if (words[j].length > 0) {
        // Extract first character and the rest of the word
        var firstChar = words[j].charAt(0);
        var restOfWord = words[j].substring(1);

        // Convert first char to uppercase and combine with rest of word
        words[j] = firstChar.toUpperCase() + restOfWord;
      }
    }

    // Join the words back together with a single space between them
    var newValue = words.join(' ');

    // Update the row data if there's been a change
    if (newValue !== originalValue) {
      row1Data[i] = newValue;
```

```
      modified = true;
      Logger.log("Cell " + (i+1) + " changed from '" + originalValue + "' to '" + newValue + "'");
    }
   }
  }

  // Write the modified data back to row 1 only if changes were made
  if (modified) {
   sheet.getRange(1, 1, 1, row1Data.length).setValues([row1Data]);
   Logger.log("First letter of each word in row 1 has been capitalized in " + sheetName + ".");
  } else {
   Logger.log("No changes were made to row 1 in " + sheetName + ".");
  }

  // Force the spreadsheet to update
  SpreadsheetApp.flush();

  return modified;
}

/**
 * Helper function to call capitalizeFirstLettersInRow1 for Sheet2
 */
function capitalizeFirstLettersInSheet2() {
  var result = capitalizeFirstLettersInRow1("Sheet2");
  if (result) {
   SpreadsheetApp.getActiveSpreadsheet().toast("Successfully capitalized row 1 in Sheet2",
"Success", 5);
  } else {
   SpreadsheetApp.getActiveSpreadsheet().toast("No changes were needed in row 1 of Sheet2",
"Information", 5);
  }
}

/**
 * Helper function to call capitalizeFirstLettersInRow1 for Sheet1
 */
function capitalizeFirstLettersInSheet1() {
  var result = capitalizeFirstLettersInRow1("Sheet1");
  if (result) {
   SpreadsheetApp.getActiveSpreadsheet().toast("Successfully capitalized row 1 in Sheet1",
"Success", 5);
  } else {
```

```
   SpreadsheetApp.getActiveSpreadsheet().toast("No changes were needed in row 1 of Sheet1",
"Information", 5);
  }
}
```

2.

We have 2 worksheets. S1 and S2
Transfer Columns A from S1 to Column F S2,
Column B S1 to Column B S2, Column C S1 to Column C S2, Column D S1 to Column E S2,
Column E S1, to Column D S2,
Column J S1 to Column G S2, Column M S1 to Column A S2, Column O S1 to Column H S2

The Code:

```
function transferColumns() {
// Open the active spreadsheet
const spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

// Get the sheets
const sheet1 = spreadsheet.getSheetByName("Sheet1");
const sheet2 = spreadsheet.getSheetByName("Sheet2");

if (!sheet1 || !sheet2) {
  throw new Error("One or both sheets not found. Ensure Sheet1 and Sheet2 exist.");
}

// Define the column mappings
const columnMappings = [
  { from: 'A', to: 'A' },
  { from: 'B', to: 'B' },
  { from: 'C', to: 'C' },
  { from: 'D', to: 'D' },
  { from: 'E', to: 'E' },
  { from: 'J', to: 'F' },
  { from: 'M', to: 'G' },
  { from: 'O', to: 'H' }
];

// Loop through the mappings and transfer data
columnMappings.forEach(mapping => {
  const lastRow = sheet1.getLastRow(); // Get the last row with data in Sheet1 for the column
  const fromRange = sheet1.getRange(`${mapping.from}1:${mapping.from}${lastRow}`);
  const values = fromRange.getValues();
```

```
    // Set the range size dynamically in Sheet2
    const toRange = sheet2.getRange(1, columnLetterToNumber(mapping.to), values.length, 1);
    toRange.setValues(values);
  });

  SpreadsheetApp.flush();
  Logger.log("Columns transferred successfully.");
}

// Helper function to convert column letters to numbers
function columnLetterToNumber(letter) {
  let column = 0;
  for (let i = 0; i < letter.length; i++) {
    column = column * 26 + (letter.charCodeAt(i) - 'A'.charCodeAt(0) + 1);
  }
  return column;
}
```

3.

Provide Separate script: In S2 starting row 2, organize the columns A to H by unique name in column A.

```
/**
 * Organizes data in sheet S2 by unique values in column A, starting from row 2.
 * This function:
 * 1. Preserves the header row (row 1)
 * 2. Sorts the data by column A values
 * 3. Groups rows with the same values in column A together
 */
function organizeS2ByUniqueNamesInColumnA() {
  // Get the active spreadsheet
  var ss = SpreadsheetApp.getActiveSpreadsheet();

  // Get the S2 sheet
  var sheet = ss.getSheetByName("S2");

  // Check if sheet exists
  if (!sheet) {
    Logger.log("Sheet 'S2' not found.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Sheet 'S2' not found", "Error", 5);
    return;
  }
```

```
  // Get the number of rows and columns with data
  var lastRow = sheet.getLastRow();
  var lastColumn = sheet.getLastColumn();

  // If there are fewer than 2 rows, there's only a header or nothing at all
  if (lastRow < 2) {
    Logger.log("Not enough data to organize. Sheet has fewer than 2 rows.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Not enough data to organize. Need at least 2
rows.", "Warning", 5);
    return;
  }

  // Get all data including headers (we'll separate them later)
  var allData = sheet.getRange(1, 1, lastRow, lastColumn).getValues();

  // Extract headers (row 1)
  var headers = allData[0];

  // Extract data rows (row 2 and beyond)
  var dataRows = allData.slice(1);

  // Create a map to store rows by their column A values
  var rowsByColumnA = {};

  // Group rows by their column A values
  for (var i = 0; i < dataRows.length; i++) {
    var row = dataRows[i];
    var columnAValue = row[0]; // Column A is index 0

    // Skip empty values in column A
    if (columnAValue === "" || columnAValue === null) {
      continue;
    }

    // Convert to string for consistent handling
    columnAValue = String(columnAValue);

    // Initialize array for this column A value if it doesn't exist
    if (!rowsByColumnA[columnAValue]) {
      rowsByColumnA[columnAValue] = [];
    }

    // Add this row to the appropriate group
```

```
    rowsByColumnA[columnAValue].push(row);
  }

  // Get unique column A values and sort them alphabetically
  var uniqueColumnAValues = Object.keys(rowsByColumnA).sort();

  // Create a new array for organized data
  var organizedData = [headers]; // Start with the headers

  // Add rows in order of sorted unique column A values
  for (var i = 0; i < uniqueColumnAValues.length; i++) {
    var columnAValue = uniqueColumnAValues[i];
    var rows = rowsByColumnA[columnAValue];

    // Add all rows for this column A value to the organized data
    for (var j = 0; j < rows.length; j++) {
      organizedData.push(rows[j]);
    }
  }

  // Clear the sheet data (except formulas and formatting)
  sheet.clearContents();

  // Write the organized data back to the sheet
  if (organizedData.length > 0) {
    sheet.getRange(1, 1, organizedData.length,
organizedData[0].length).setValues(organizedData);
    Logger.log("Data in S2 has been organized by unique values in column A.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Data in S2 has been organized by unique
values in column A.", "Success", 5);
  } else {
    Logger.log("No data to organize.");
    SpreadsheetApp.getActiveSpreadsheet().toast("No data to organize.", "Warning", 5);
  }

  // Force the spreadsheet to update
  SpreadsheetApp.flush();
}
```

4.
Read sheet S2 and by unique values in column A,
insert empty rows between different groups, and stop program at two consecutive empty rows.
Handle ALL rows in the spreadsheet without arbitrary limits.

```
/**
 * Fixed function that organizes data in sheet S2 by unique values in column A,
 * inserts empty rows between different groups, and stops at two consecutive empty rows.
 * This version handles ALL rows in the spreadsheet without arbitrary limits.
 */
function separateGroupsWithEmptyRowsFixed() {
  // Start timing the execution
  var startTime = new Date();

  // Get the active spreadsheet and the S2 sheet
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getSheetByName("S2");

  // Check if sheet exists
  if (!sheet) {
    Logger.log("Sheet 'S2' not found.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Sheet 'S2' not found", "Error", 5);
    return;
  }

  // Get the actual number of rows and columns with data - NO LIMITS
  var lastRow = sheet.getLastRow();
  var lastColumn = sheet.getLastColumn();

  // Log for debugging
  Logger.log("Processing sheet with " + lastRow + " rows and " + lastColumn + " columns");

  // If there are fewer than 2 rows, there's only a header or nothing at all
  if (lastRow < 2) {
    Logger.log("Not enough data to organize. Sheet has fewer than 2 rows.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Not enough data to organize. Need at least 2
rows.", "Warning", 5);
    return;
  }

  // Get data in batches if there are many rows to avoid memory issues
  var allData = [];
  var batchSize = 1000; // Process 1000 rows at a time, but don't limit total rows
```

```
  // Get header row separately
  var headers = sheet.getRange(1, 1, 1, lastColumn).getValues()[0];
  allData.push(headers);

  // Process data in batches for better memory management
  for (var i = 2; i <= lastRow; i += batchSize) {
    var rowsToGet = Math.min(batchSize, lastRow - i + 1);
    var batchData = sheet.getRange(i, 1, rowsToGet, lastColumn).getValues();
    allData = allData.concat(batchData);

    // Yield to prevent timeout on very large datasets
    if (rowsToGet === batchSize) {
      SpreadsheetApp.flush();
      Logger.log("Processed rows " + i + " to " + (i + rowsToGet - 1));
    }
  }

  // Check for two consecutive empty rows to find where to stop processing
  var stopAtIndex = -1;
  for (var i = 1; i < allData.length - 1; i++) {
    if (isEmptyRow(allData[i]) && isEmptyRow(allData[i+1])) {
      stopAtIndex = i;
      Logger.log("Found two consecutive empty rows at positions " + (i+1) + " and " + (i+2) + ". Will
stop processing here.");
      break;
    }
  }

  // If we found two consecutive empty rows, limit our processing
  var processedData;
  if (stopAtIndex > 0) {
    processedData = allData.slice(1, stopAtIndex);
    Logger.log("Stopping at row " + (stopAtIndex + 1) + " due to two consecutive empty rows");
  } else {
    processedData = allData.slice(1);
  }

  // Group the data by column A values
  var groups = {};
  var uniqueValues = [];

  for (var i = 0; i < processedData.length; i++) {
    var row = processedData[i];
```

```
  // Skip empty rows
  if (isEmptyRow(row)) continue;

  var columnAValue = String(row[0] || "");

  // First time seeing this value?
  if (!groups[columnAValue]) {
    groups[columnAValue] = [];
    uniqueValues.push(columnAValue);
  }

  // Add this row to its group
  groups[columnAValue].push(row);
}

// Sort unique values alphabetically for consistent output
uniqueValues.sort();

// Create new worksheet data with headers and grouped data
var newData = [headers]; // Start with headers

// Add each group's data with empty rows between groups
for (var i = 0; i < uniqueValues.length; i++) {
  var value = uniqueValues[i];
  var groupRows = groups[value];

  // Add all rows for this group
  for (var j = 0; j < groupRows.length; j++) {
    newData.push(groupRows[j]);
  }

  // Add an empty row after the group (except for the last group)
  if (i < uniqueValues.length - 1) {
    newData.push(Array(lastColumn).fill(""));
  }
}

// Apply the changes in batches to handle large datasets
try {
  // Clear the sheet first
  sheet.clear();

  // Write the header row
  sheet.getRange(1, 1, 1, headers.length).setValues([headers]);
```

```
    // Write data in batches if there's a lot
    if (newData.length > 1) {
      var dataToWrite = newData.slice(1); // Skip header as we've already written it

      // Set maximum batch size for writing
      var writeBatchSize = 1000;

      for (var i = 0; i < dataToWrite.length; i += writeBatchSize) {
        var batchToWrite = dataToWrite.slice(i, i + writeBatchSize);
        if (batchToWrite.length > 0) {
          // row index is i+2 because we're 0-indexed, and already wrote row 1 (header)
          sheet.getRange(i + 2, 1, batchToWrite.length, lastColumn).setValues(batchToWrite);
          SpreadsheetApp.flush(); // Force update to avoid timeout
          Logger.log("Wrote rows " + (i + 2) + " to " + (i + batchToWrite.length + 1));
        }
      }
    }

    // Success message
    var executionTime = (new Date() - startTime) / 1000;
    var rowsProcessed = stopAtIndex > 0 ? stopAtIndex : processedData.length;

    Logger.log("Data organization completed in " + executionTime + " seconds. Processed " +
rowsProcessed + " rows.");
    SpreadsheetApp.getActiveSpreadsheet().toast(
      "Data organized with empty rows between groups. Processed " + rowsProcessed + " rows in
" + executionTime + " seconds",
      "Success",
      8
    );
  } catch (e) {
    Logger.log("Error: " + e.toString());
    SpreadsheetApp.getActiveSpreadsheet().toast(
      "Error organizing data: " + e.toString(),
      "Error",
      10
    );
  }
}

/**
 * Faster helper function to check if a row is empty
 */
```

```
function isEmptyRow(row) {
  for (var i = 0; i < row.length; i++) {
    // Check if cell has any content
    if (row[i] !== "" && row[i] !== null) {
      return false;
    }
  }
  return true;
}
```

5.
Fill blank cells in column H with an 'x' character in sheet S2,
starting from row 2. Function should preserve all other data.

```
 /**
 * Fills blank cells in column H with an 'x' character in sheet S2,
 * starting from row 2. This function preserves all other data.
 */
function fillBlankCellsInColumnH() {
  // Start timing the execution
  var startTime = new Date();

  // Get the active spreadsheet and the S2 sheet
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getSheetByName("S2");

  // Check if sheet exists
  if (!sheet) {
    Logger.log("Sheet 'S2' not found.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Sheet 'S2' not found", "Error", 5);
    return;
  }

  // Get the number of rows with data
  var lastRow = sheet.getLastRow();

  // If there are fewer than 2 rows, there's only a header or nothing at all
  if (lastRow < 2) {
    Logger.log("Not enough data. Sheet has fewer than 2 rows.");
    SpreadsheetApp.getActiveSpreadsheet().toast("Not enough data. Need at least 2 rows.",
"Warning", 5);
    return;
```

```
  }

  // Column H is the 8th column (index 7 in 0-based arrays)
  var columnHIndex = 8;

  // Process data in batches to avoid memory issues
  var batchSize = 1000; // Process 1000 rows at a time
  var cellsModified = 0;

  try {
    // Process the sheet in batches
    for (var startRow = 2; startRow <= lastRow; startRow += batchSize) {
      // Calculate how many rows to process in this batch
      var numRows = Math.min(batchSize, lastRow - startRow + 1);

      // Get the data for column H in this batch
      var range = sheet.getRange(startRow, columnHIndex, numRows, 1);
      var values = range.getValues();

      // Track which cells need to be modified
      var cellsToModify = [];

      // Check each cell in column H
      for (var i = 0; i < values.length; i++) {
        // If the cell is blank (empty or null)
        if (values[i][0] === "" || values[i][0] === null) {
          // Change the value to 'x'
          values[i][0] = "x";

          // Add to the list of cells that were modified
          cellsToModify.push(startRow + i);
          cellsModified++;
        }
      }

      // Write the updated values back to the sheet
      range.setValues(values);

      // Log progress
      if (cellsToModify.length > 0) {
        Logger.log("Modified " + cellsToModify.length + " cells in rows " + startRow + " to " +
(startRow + numRows - 1));
      }
```

```
      // Yield to prevent timeout on very large datasets
      if (numRows === batchSize) {
        SpreadsheetApp.flush();
      }
    }

    // Success message
    var executionTime = (new Date() - startTime) / 1000;

    Logger.log("Added 'x' to " + cellsModified + " blank cells in column H, completed in " +
executionTime + " seconds.");
    SpreadsheetApp.getActiveSpreadsheet().toast(
      "Added 'x' to " + cellsModified + " blank cells in column H in " + executionTime + " seconds",
      "Success",
      5
    );
  } catch (e) {
    Logger.log("Error: " + e.toString());
    SpreadsheetApp.getActiveSpreadsheet().toast(
      "Error filling blank cells: " + e.toString(),
      "Error",
      10
    );
  }
}
```

6.
Replace 'x' characters in column H of sheet S2 with the sum of numeric values in the cells above each 'x' Format Values in bold red text.

```
function replaceXWithSumOfValuesAbove() {
// Start timing the execution
var startTime = new Date();

// Get the active spreadsheet and the S2 sheet
var ss = SpreadsheetApp.getActiveSpreadsheet();
var sheet = ss.getSheetByName("S2");

// Check if sheet exists
if (!sheet) {
  Logger.log("Sheet 'S2' not found.");
  SpreadsheetApp.getActiveSpreadsheet().toast("Sheet 'S2' not found", "Error", 5);
  return;
}

// Get the number of rows with data
var lastRow = sheet.getLastRow();

// If there are fewer than 2 rows, there's only a header or nothing at all
if (lastRow < 2) {
  Logger.log("Not enough data. Sheet has fewer than 2 rows.");
  SpreadsheetApp.getActiveSpreadsheet().toast("Not enough data. Need at least 2 rows.",
"Warning", 5);
  return;
}

// Column H is the 8th column
var columnHIndex = 8;

try {
  // Get all values from column H (excluding header row)
  var columnHRange = sheet.getRange(2, columnHIndex, lastRow - 1, 1);
  var columnHValues = columnHRange.getValues();

  // Debug log to check if we're getting data correctly
  Logger.log("Retrieved " + columnHValues.length + " values from column H");

  // Track cells to update and their indices
  var cellsToUpdate = [];
  var cellIndices = [];
```

```
  // Track running sum of numeric values
  var runningSum = 0;

  // Process each cell in column H
  for (var i = 0; i < columnHValues.length; i++) {
    var currentValue = columnHValues[i][0];

    // Debug log for special values to help troubleshoot
    if (currentValue === 'x') {
      Logger.log("Found 'x' at row " + (i + 2) + ", current running sum: " + runningSum);
    }

    // If current cell contains an 'x' (ensuring case sensitivity and exact match)
    if (currentValue === 'x') {
      // Add this cell to the update list with the current running sum
      cellsToUpdate.push([runningSum]);
      cellIndices.push(i + 2); // +2 because we're starting from row 2 and array is 0-indexed
    }
    // Otherwise, if it's a number, add it to our running sum
    else if (typeof currentValue === 'number') {
      runningSum += currentValue;
      Logger.log("Added number: " + currentValue + ", new running sum: " + runningSum);
    }
    // Try to convert strings to numbers (in case they're formatted as text)
    else if (typeof currentValue === 'string') {
      // Remove any commas, currency symbols, etc.
      var cleanValue = currentValue.replace(/[$,]/g, '');
      var numValue = parseFloat(cleanValue);

      if (!isNaN(numValue)) {
        runningSum += numValue;
        Logger.log("Converted string '" + currentValue + "' to number: " + numValue + ", new
running sum: " + runningSum);
      }
    }
  }

  // Log summary before updating
  Logger.log("Found " + cellsToUpdate.length + " 'x' values to replace");

  // Update all cells with 'x' to their respective sums - using batch update for efficiency
  if (cellsToUpdate.length > 0) {
    // Update cells in batches of 100 for better performance
```

```
    var batchSize = 100;

    for (var i = 0; i < cellIndices.length; i += batchSize) {
      var batchIndicesLength = Math.min(batchSize, cellIndices.length - i);
      var batchValues = cellsToUpdate.slice(i, i + batchIndicesLength);

      // Update each cell in this batch individually as they may not be contiguous
      for (var j = 0; j < batchIndicesLength; j++) {
        var rowIndex = cellIndices[i + j];
        var cellRange = sheet.getRange(rowIndex, columnHIndex);

        // Set the value
        cellRange.setValue(batchValues[j][0]);

        // Apply bold red formatting to the cell
        cellRange.setFontWeight("bold");
        cellRange.setFontColor("#FF0000");
      }

      // Yield occasionally to prevent timeout
      if (i > 0) {
        SpreadsheetApp.flush();
        Logger.log("Processed " + Math.min(i + batchIndicesLength, cellIndices.length) + "
replacements so far");
      }
    }
  }

  // Success message
  var executionTime = (new Date() - startTime) / 1000;

  Logger.log("Replaced " + cellsToUpdate.length + " 'x' values with bold red sums in column H,
completed in " + executionTime + " seconds.");
  SpreadsheetApp.getActiveSpreadsheet().toast(
    "Replaced " + cellsToUpdate.length + " 'x' values with bold red sums in column H in " +
executionTime + " seconds",
    "Success",
    5
  );
 } catch (e) {
  Logger.log("Error: " + e.toString());
  SpreadsheetApp.getActiveSpreadsheet().toast(
    "Error replacing 'x' values: " + e.toString(),
    "Error",
```

```
    10
  );
 }
}
```

7. Run All.

```
function executeAllScripts() {
 var functionOrder = [
  'capitalizeFirstLettersInRow1(sheetName)', // this is the function first page
  'transferDataFromS1ToS2AltMapping', // this is the function name of your second page
  'organizeS2ByUniqueNamesInColumnA', //.....third page
  'separateGroupsWithEmptyRowsFixed', //......fourth page
  'fillBlankCellsInColumnH',         //....etc....
  'replaceXWithSumOfValuesAbove'     //....etc...
 ];

 try {
  Logger.log('Starting script execution: ' + new Date());

  for (var i = 0; i < functionOrder.length; i++) {
   var functionName = functionOrder[i];

   if (typeof this[functionName] === 'function') {
    Logger.log('Executing: ' + functionName);

    // Handle special case for first function which requires a parameter
    if (functionName === 'capitalizeFirstLettersInRow1') {
     this[functionName]('S2'); // Assuming 'S2' is the target sheet name
    } else {
     this[functionName]();
    }

    Logger.log(functionName + ' completed');
   } else {
    Logger.log('Warning: Function ' + functionName + ' not found');
   }
  }

  Logger.log('All scripts completed successfully: ' + new Date());
  SpreadsheetApp.getActiveSpreadsheet().toast('All scripts executed successfully!');

 } catch (error) {
  Logger.log('Error: ' + error.toString());
```

```
    SpreadsheetApp.getActiveSpreadsheet().toast('Error: ' + error.toString(), 'Script Error', 10);
  }
}

/**
 * Creates a custom menu when the spreadsheet is opened
 */
function onOpen() {
  SpreadsheetApp.getUi()
    .createMenu('Execute Scripts')
    .addItem('Run All Scripts', 'executeAllScripts')
    .addToUi();
}
```