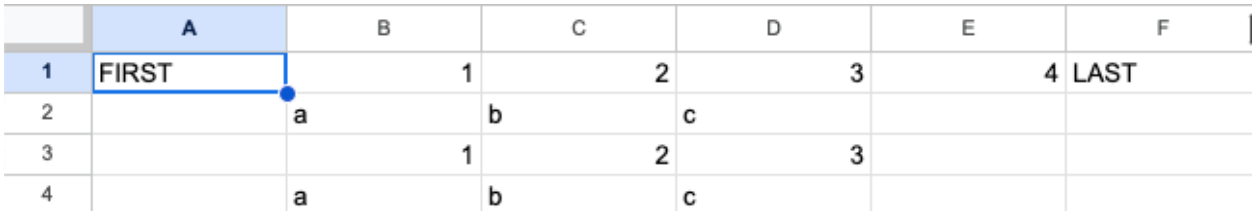
Google Apps Script Video Lessons Source Code

Add Columns with Apps Script 1
Sort By Column Index value 2
Filter the rows in a Google Sheet 4
Create a pivot table in a Google Sheets 6
Create a new Google Slides presentation 9

Add Columns with Apps Script

https://youtu.be/B_2HhWygE8g



	A	B	C	D	E	F
1	FIRST	1	2	3	4	LAST
2		a	b	c		
3		1	2	3		
4		a	b	c		

```
function addColumn() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheets()[0];
  const sheetName = sheet.getName();
  sheet.insertColumnBefore(1);
  const lastCol = sheet.getMaxColumns();
  sheet.insertColumnAfter(lastCol);
  sheet.getRange(1,1).setValue('FIRST');
  sheet.getRange(1,lastCol+1).setValue('LAST');
  Logger.log(lastCol);
}
```

This code defines a Google Apps Script function called "addColumn()" which adds a new column to the left and right of an existing spreadsheet.

Here's what each line of the code does:

const ss = SpreadsheetApp.getActiveSpreadsheet();

1. This line gets the currently active Google Spreadsheet and assigns it to the variable ss.

const sheet = ss.getSheets()[0];

2. This line gets the first sheet in the spreadsheet and assigns it to the variable sheet.

const sheetName = sheet.getName();

3. This line gets the name of the sheet and assigns it to the variable sheetName.

sheet.insertColumnBefore(1);

4. This line inserts a new column to the left of the first column of the sheet.

const lastCol = sheet.getMaxColumns();

5. This line gets the number of columns in the sheet and assigns it to the variable lastCol.

sheet.insertColumnAfter(lastCol);

6. This line inserts a new column to the right of the last column of the sheet.

sheet.getRange(1,1).setValue('FIRST');

7. This line sets the value of the first cell (1,1) in the sheet to the string "FIRST".

sheet.getRange(1,lastCol+1).setValue('LAST');

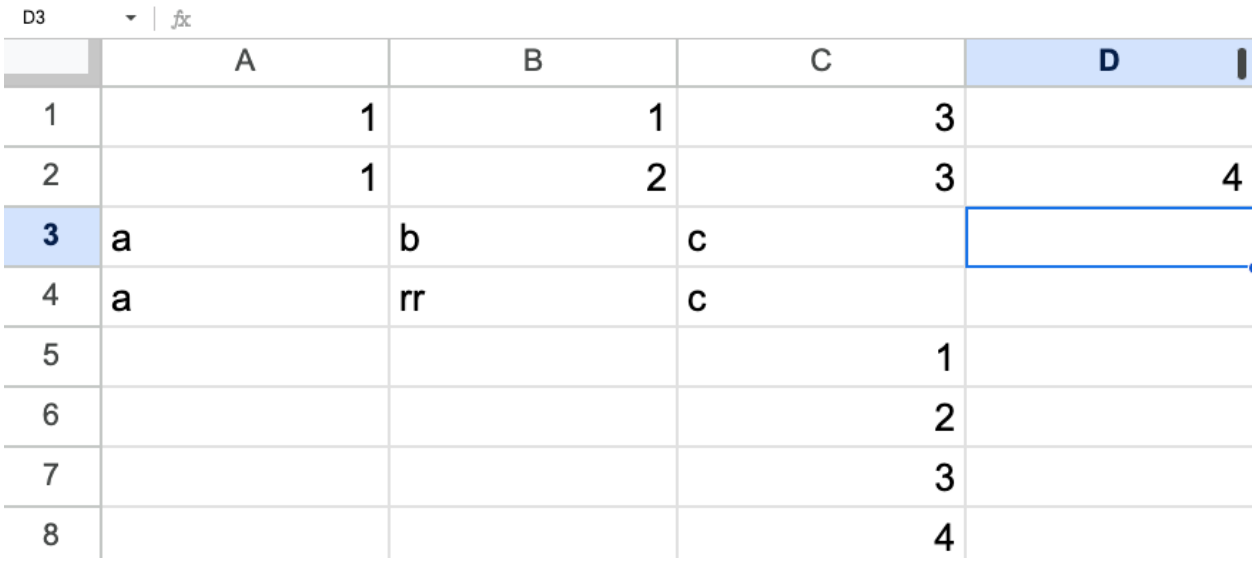
8. This line sets the value of the last cell in the new column (1, lastCol+1) to the string "LAST".

Logger.log(lastCol);

9. This line logs the value of lastCol to the console for debugging purposes.

Sort By Column Index value

https://youtu.be/zV2WBghVWOM



	A	B	C	D
1	1	1	3	
2	1	2	3	4
3	a	b	c	
4	a	rr	c	
5			1	
6			2	
7			3	
8			4	

```
function sortRows(){
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getActiveSheet();
  Logger.log(sheet.getSheetName());
  const sortCol = 1;
  sheet.sort(sortCol);
}
```

This is a Google Apps Script function that sorts the rows in the currently active sheet based on the values in the first column.
Here's what each line of the code does:
const ss = SpreadsheetApp.getActiveSpreadsheet();

1. This line gets the currently active Google Spreadsheet and assigns it to the variable ss.

const sheet = ss.getActiveSheet();

2. This line gets the currently active sheet in the spreadsheet and assigns it to the variable sheet.

Logger.log(sheet.getSheetName());

3. This line logs the name of the sheet to the console for debugging purposes.

const sortCol = 1;

4. This line assigns the value 1 to the variable sortCol, which will be used as the sort column.

sheet.sort(sortCol);

5. This line sorts the rows in the sheet based on the values in the first column (column A), which is specified by the sortCol variable. By default, the sort order is ascending, but you can change it to descending by passing true as a second argument to the sort() method.

After running this function, the rows in the active sheet will be rearranged so that the values in the first column are in ascending order.

Filter the rows in a Google Sheet

https://youtu.be/IXtDhmtMU-M

Example of how to use SpreadsheetApp.newFilterCriteria() in Google Apps Script to create a new filter criteria object:

No Filter Applied

	A	B	C
1	Column 1	Column 2	Column 3
2	apple	red	round
3	banana	yellow	curved
4	orange	orange	round
5	grape	purple	round
6	peach	orange	fuzzy
7	kiwi	green	fuzzy
8	apple	green	fuzzy
9	test	test	apple

With Filter Applied

	A	B	C
1	Column 1	Column 2	Column 3
3	banana	yellow	curved
5	grape	purple	round
6	peach	orange	fuzzy
7	kiwi	green	fuzzy
9	test	test	apple

Data for Sheet Values

Column 1	Column 2	Column 3
apple	red	round
banana	yellow	curved
orange	orange	round
grape	purple	round
peach	orange	fuzzy
kiwi	green	fuzzy

```
function createFilter() {
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheetByName("Sheet1");
  const range = sheet.getDataRange();

  const criteria = SpreadsheetApp.newFilterCriteria()
    .setHiddenValues(['apple', 'orange'])
    .build();
```

```
  const filter = range.getFilter() || range.createFilter();
  filter.setColumnFilterCriteria(1, criteria);
}
```

This function creates a new filter criteria object using SpreadsheetApp.newFilterCriteria() and sets the criteria to hide rows containing the values "apple" or "orange" in the first column of the sheet named "Sheet1".
Here's a breakdown of the code:

- const ss = SpreadsheetApp.getActiveSpreadsheet();: gets the active spreadsheet.
- const sheet = ss.getSheetByName("Sheet1");: gets the sheet named "Sheet1".
- const range = sheet.getDataRange();: gets the data range of the sheet.
- const criteria = SpreadsheetApp.newFilterCriteria(): creates a new filter criteria object.
- .setHiddenValues(['apple', 'orange']): sets the hidden values for the criteria to an array of values.
- .build();: builds the criteria object.
- const filter = range.getFilter() || range.createFilter();: gets the filter of the range, or creates a new one if it doesn't exist.
- filter.setColumnFilterCriteria(1, criteria);: sets the criteria for the first column of the range.

This example demonstrates how to use SpreadsheetApp.newFilterCriteria() to create a filter criteria object and set it on a sheet range to filter out rows containing specific values in a certain column. You can customize this code to create different filter criteria based on your needs.

Create a pivot table in a Google Sheets

https://youtu.be/wdNNiDTxaaI

Pivot Table sheet data

	A	B
1	*Name*	COUNTA of Name
2	Alice	3
3	Bob	3
4	Eve	3
5	**Grand Total**	**9**

Pivot Table Source Data

	A	B	C	D
1	1	Name	Email	Phone
2	2	Alice	a@a.com	123456
3	3	Bob	b@b.com	234567
4	4	Alice	a@a.com	345678
5	5	Bob	b@b.com	456789
6	6	Eve	e@e.com	567890
7	7	Alice	a@a.com	678901
8	8	Eve	e@e.com	789012
9	9	Eve	e@e.com	890123
10	10	Bob	b@b.com	901234

Data for the example code in a table format:

	A	B	C
1	Name	Email	Phone
2	Alice	a@a.com	123456
3	Bob	b@b.com	234567

4	Alice	a@a.com	345678
5	Bob	b@b.com	456789
6	Eve	e@e.com	567890
7	Alice	a@a.com	678901
8	Eve	e@e.com	789012
9	Eve	e@e.com	890123
10	Bob	b@b.com	901234

This table has three columns: "Name", "Email", and "Phone". The first row contains the column headers. The data in this table shows the names, email addresses, and phone numbers of several people. The example code creates a pivot table based on this data, with the column headers as "Name", "Email", and "Phone", the row header as "Country", and the value as the count of each person's occurrences in the data.

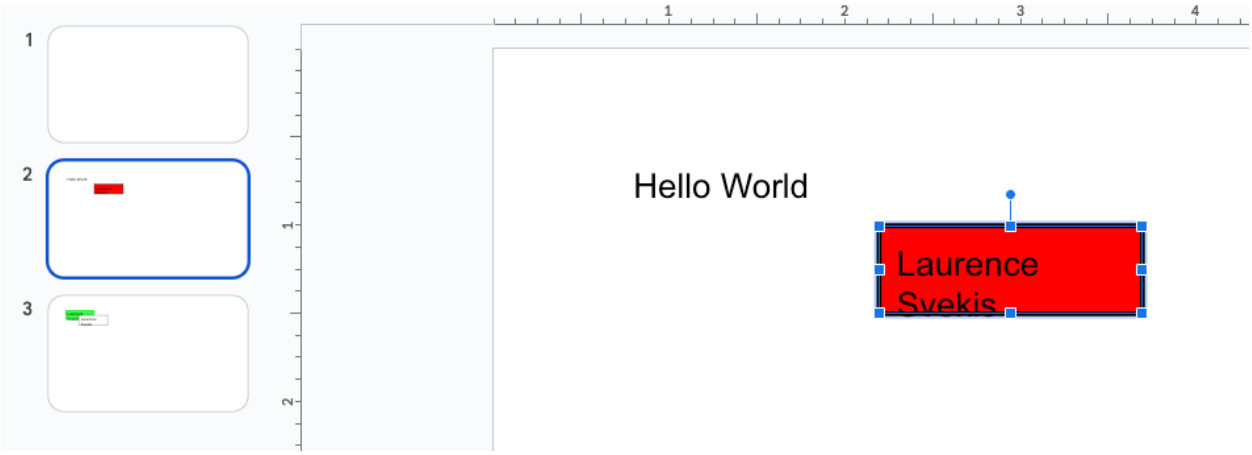
```
function createPivot(){
  const ss = SpreadsheetApp.getActiveSpreadsheet();
  const sheet = ss.getSheetByName('Sheet3');
  const psheet = ss.insertSheet();
  const pivotTable =
psheet.getRange("A1").createPivotTable(sheet.getDataRange());
  pivotTable.addRowGroup(2);
  pivotTable.addPivotValue(2,
SpreadsheetApp.PivotTableSummarizeFunction.COUNTA);
}
```

The code creates a pivot table in a new sheet named psheet based on the data in Sheet3 of the active spreadsheet. The createPivotTable() method is used to create the pivot table in cell A1 of the psheet. The row groups are set by calling addRowGroup(2) with column 2 as the argument, and the

pivot value is set to COUNTA for column 2 by calling addPivotValue(2, SpreadsheetApp.PivotTableSummarizeFunction.COUNTA).
Overall, the code creates a simple pivot table in a new sheet that groups data by the values in column 2 of Sheet3 and calculates the count of non-empty cells for each group.

Create a new Google Slides presentation

https://youtu.be/DhVNHaR2X1E




```
function createNewSlidePresentation(){
  const slideName = 'Presetation 1';
  const presentation = SlidesApp.create(slideName);
  Logger.log(presentation.getId());
  const slide1 =
presentation.appendSlide(SlidesApp.PredefinedLayout.BLANK);
  const textBox =
slide1.insertShape(SlidesApp.ShapeType.TEXT_BOX);
  textBox.getText().setText('Hello World');
  textBox.setTop(40).setLeft(50);
  const subtitle = slide1.insertTextBox('Laurence Svekis');
  subtitle.getText().setText('Laurence Svekis');
  subtitle.getBorder().setWeight(2);
  subtitle.getFill().setSolidFill('#ff0000');
  subtitle.setLeft(400).setTop(20);
```

```
  const slide2 =
presentation.appendSlide(SlidesApp.PredefinedLayout.TITLE);
  const newTitle =
slide2.insertShape(SlidesApp.ShapeType.TEXT_BOX);
  newTitle.getText().setText('Laurence Svekis');
  newTitle.setTop(40).setLeft(50);
  newTitle.getFill().setSolidFill('#00ff00');

  const newBody =
slide2.insertShape(SlidesApp.ShapeType.TEXT_BOX);
  newBody.getText().setText('Laurence Svekis');
  newBody.getBorder().setWeight(1);
  newBody.getFill().setSolidFill('#ffffff');
  newBody.setLeft(100).setTop(60);
}
```

The code createNewSlidePresentation() creates a new Google Slides presentation and adds two slides to it. The first slide is a blank layout and the second slide has a title layout.

In more detail, the code first sets the name of the presentation to "Presentation 1" and creates the presentation using SlidesApp.create(slideName). The presentation ID is then logged to the console using Logger.log(presentation.getId()).

Next, the code appends a new slide to the presentation using presentation.appendSlide(SlidesApp.PredefinedLayout.BLANK), which returns a Slide object. The Slide object is assigned to slide1.

On slide1, the code inserts a text box using slide1.insertShape(SlidesApp.ShapeType.TEXT_BOX). The text inside the text box is set to "Hello World" using textBox.getText().setText('Hello World'), and the position of the text box is set to textBox.setTop(40).setLeft(50).

The code also inserts a subtitle text box on slide1 using slide1.insertTextBox('Laurence Svekis'). The text inside the subtitle is set to "Laurence Svekis" using subtitle.getText().setText('Laurence Svekis'), the border of the subtitle is set to have a weight of 2 using

subtitle.getBorder().setWeight(2), the fill color of the subtitle is set to red using subtitle.getFill().setSolidFill('#ff0000'), and the position of the subtitle is set to subtitle.setLeft(400).setTop(20).

The code then appends a second slide to the presentation using presentation.appendSlide(SlidesApp.PredefinedLayout.TITLE), which returns a Slide object. The Slide object is assigned to slide2.

On slide2, the code inserts a title text box using slide2.insertShape(SlidesApp.ShapeType.TEXT_BOX). The text inside the title text box is set to "Laurence Svekis" using newTitle.getText().setText('Laurence Svekis'), the fill color of the title text box is set to green using newTitle.getFill().setSolidFill('#00ff00'), and the position of the title text box is set to newTitle.setTop(40).setLeft(50).

Finally, the code inserts a body text box on slide2 using slide2.insertShape(SlidesApp.ShapeType.TEXT_BOX). The text inside the body text box is set to "Laurence Svekis" using newBody.getText().setText('Laurence Svekis'), the border of the body text box is set to have a weight of 1 using newBody.getBorder().setWeight(1), the fill color of the body text box is set to white using newBody.getFill().setSolidFill('#ffffff'), and the position of the body text box is set to newBody.setLeft(100).setTop(60).