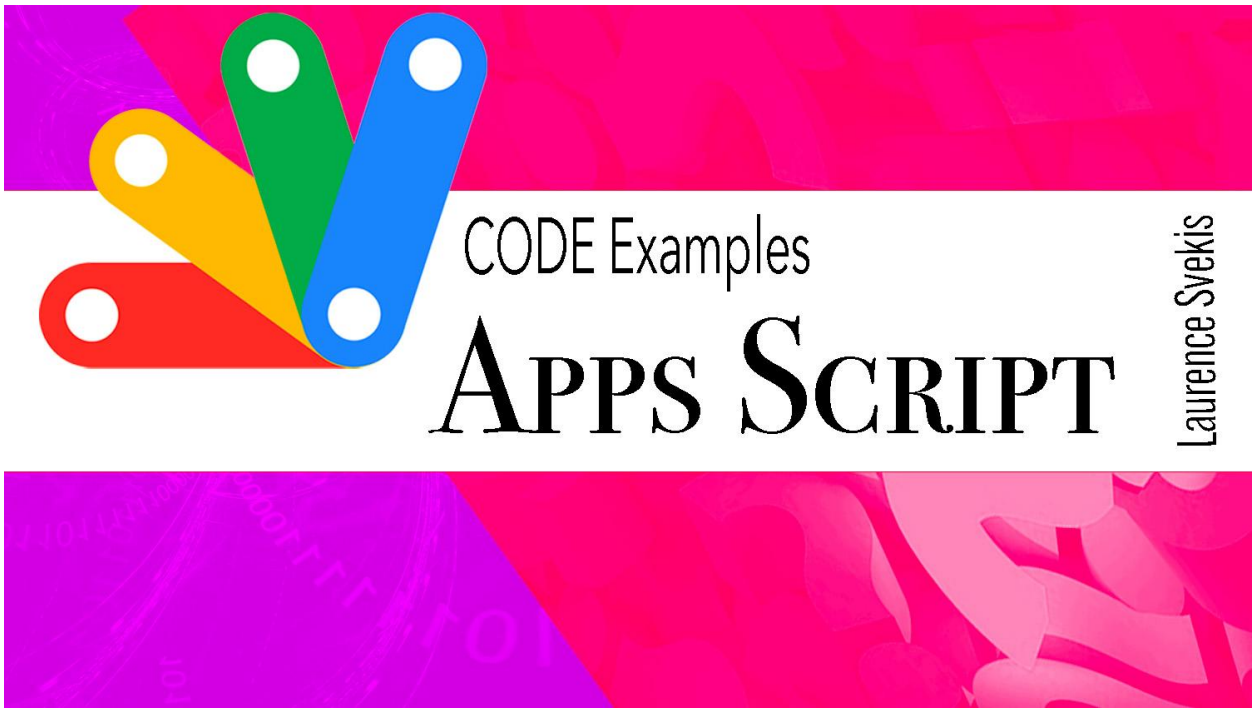
Apps Script Coding Examples V2

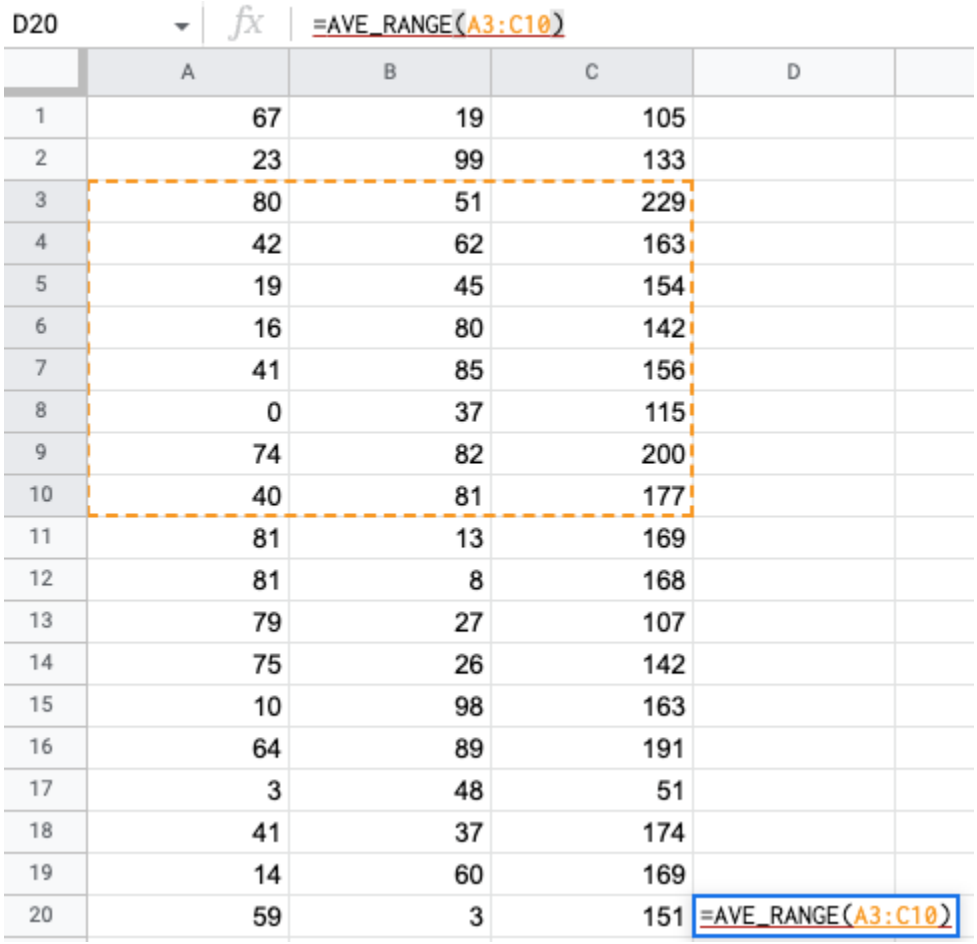


Create a custom function to calculate the average of a range of cells:

```
function AVERAGE_RANGE(range) {
  var sum = 0;
  var count = 0;
  for (var i = 0; i < range.length; i++) {
    for (var j = 0; j < range[i].length; j++) {
      sum += range[i][j];
      count++;
    }
  }
  return sum / count;
}
```

Explanation: This code defines a custom function called AVERAGE_RANGE that takes a range of cells as input and calculates the average value of those cells. The function uses a

nested loop to iterate through each cell in the range, adding up the values and counting the number of cells. It then returns the average value by dividing the sum by the count.




```
function AVE_RANGE(range){
  let sum = 0;
  let count = 0;
  for(let row=0;row<range.length;row++){
    for(let col=0;col<range[row].length;col++){
      sum += range[row][col];
      count++;
    }
  }
```

```
  return Math.ceil(sum/count);
}
```

Custom Sheet Formula

https://youtu.be/XARTblSMQ-I

The AVE_RANGE function takes in a 2D array range, which represents a rectangular range of cells in a spreadsheet. It calculates the average value of all the cells in the range and rounds it up to the nearest integer.

To do this, the function initializes two variables, sum and count, to 0. It then iterates over each cell in the range using a nested for loop. For each cell, it adds its value to sum and increments count.

After all the cells have been processed, the function divides sum by count to calculate the average value of the cells. It then uses the Math.ceil() function to round this value up to the nearest integer, and returns the result.

Overall, this function can be useful in a variety of contexts where you need to calculate the average value of a range of cells, such as in data analysis or financial modeling.

Automatically send an email reminder based on a specific date:

```
function sendReminderEmail() {
```

```
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("T
asks");
  var data = sheet.getDataRange().getValues();
  var today = new Date();
  for (var i = 1; i < data.length; i++) {
    var date = new Date(data[i][2]);
    var diff = Math.round((date - today) / (1000 * 60 *
60 * 24));
    if (diff === 1) {
      var email = data[i][3];
      var subject = "Reminder: " + data[i][1];
      var body = "This is a reminder that your task \""
+ data[i][1] + "\" is due tomorrow.";
      MailApp.sendEmail(email, subject, body);
    }
  }
}
```

Explanation: This code defines a function called sendReminderEmail that retrieves data from a Google Sheet and sends an email reminder to users when their task is due the next day. The function calculates the number of days between today's date and the due date for each task, and sends an email if the difference is 1 day.

Send email reminder Automation

https://youtu.be/b5GTk3taFqQ

```
function senderEmail(){
  const sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName('U
sers');
  const data = sheet.getDataRange().getValues();
  const users = data.slice(1);
  const today = new Date();
  users.forEach((user,index) => {
    if(user[4]==''){
      const date = new Date(user[3]);
      const diff =
Math.round((date-today)/1000*60*60*24);
      const range = sheet.getRange(index+2,5);
      if(diff<0){
        const email = user[1];
        const subject = 'Passed';
        const body = 'This is a reminder';
        MailApp.sendEmail(email,subject,body);
        range.setValue(today);
      }
      Logger.log(diff);
    }else{
      Logger.log(user[4]);
    }
  })
  //Logger.log(users);
}
```

The senderEmail function retrieves data from a Google Sheet named "Users", which contains information about users including their email addresses, a date of last contact, and a reminder flag.

The function then iterates over each row of user data and checks if the reminder flag is empty.

If the reminder flag is empty, the function calculates the difference between today's date and the date of last contact for the user. If the difference is negative (meaning the last contact date is in the past), the function sends an email to the user reminding them to make contact, and updates the reminder flag with today's date.

The function also logs the difference in days between today's date and the last contact date for each user. If the reminder flag is not empty, the function simply logs its value.

Overall, this function appears to be designed to automate the process of sending reminders to users who have not made contact in a while, and to track when those reminders were sent. However, without more context about the purpose and use case of the "Users" sheet, it is difficult to determine the full scope and functionality of the code.

Create a custom menu in Google Sheets to run a script:

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom Menu')
      .addItem('Run Script', 'myFunction')
      .addToUi();
}

function myFunction() {
```

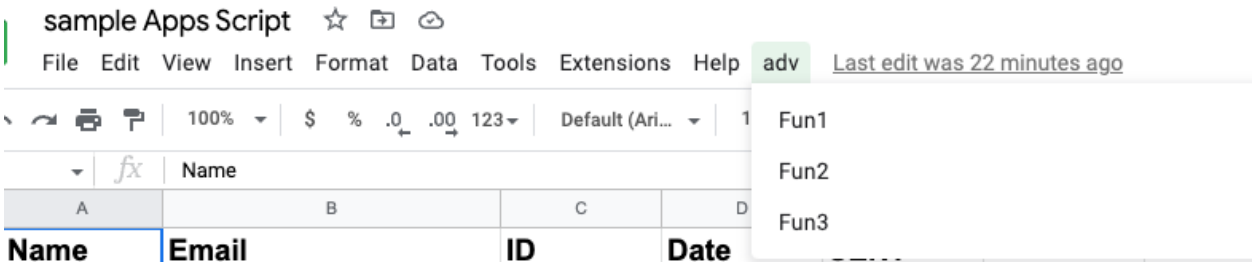
```
  // your code here
}
```

Explanation: This code creates a custom menu in Google Sheets called "Custom Menu" that contains an option to run a script called myFunction. The onOpen function is a built-in function that runs automatically when the Google Sheet is opened, and it creates the custom menu using the Ui class. The myFunction function can be replaced with any script you want to run from the menu.

Add and run functions.

Custom Sheet UI Menu

https://youtu.be/SVpLYhDv8UQ




```
function onOpen(){
  const ui = SpreadsheetApp.getUi();
  ui.createMenu('adv')
    .addItem('Fun1','fun1')
    .addItem('Fun2','fun2')
    .addItem('Fun3','fun3')
    .addToUi()
}

function fun1(){
```

```
  SpreadsheetApp.getUi().alert('You clicked 1');
}
function fun2(){
  SpreadsheetApp.getUi().alert('You clicked 2');
}
function fun3(){
  SpreadsheetApp.getUi().alert('You clicked 3');
}
```

The above code defines four functions - onOpen, fun1, fun2, and fun3 - that can be used in conjunction with a Google Sheets document.

The onOpen function is a special function that is called automatically when the sheet is opened. It creates a new custom menu in the sheet's UI called "adv", which contains three menu items - "Fun1", "Fun2", and "Fun3" - each of which calls one of the other three functions when clicked. The addToUi method is then called to add this menu to the sheet's UI.

The fun1, fun2, and fun3 functions are each simple functions that display an alert message using the SpreadsheetApp.getUi().alert() method. When one of the menu items is clicked, the corresponding function is called, which displays an alert message with a unique message indicating which function was triggered.

Overall, this code creates a simple custom menu in a Google Sheets document, which provides easy access to a few custom functions that can perform various actions or display information to the user. This can be particularly useful for creating custom workflows or automating common tasks within a spreadsheet.

Use the Google Drive API to create a new folder:

```
function createFolder() {
  var folderName = "New Folder";
  var folder = DriveApp.createFolder(folderName);
  Logger.log("Created folder with ID: " +
folder.getId());
}
```

Explanation: This code defines a function called createFolder that uses the Google Drive API to create a new folder with the name "New Folder". The function calls the createFolder method of the DriveApp class, which returns a folder object with information about the new folder. The Logger.log method is used to print the ID of the new folder to the Logs.

Create new Folder within Folder

https://youtu.be/D3cxVxWrzWU

```
function cFolder(){
  const folderName = 'New Folder Today';
  const folder = DriveApp.createFolder(folderName);
  Logger.log(`Created Folder ID ${folder.getId()}`);
}

function addFolder(){
  const id = '1SjwbFJA5FvHXpNQMSE';
  const folderName = 'Inside Folder';
  const main = DriveApp.getFolderById(id);
  const folder = main.createFolder(folderName);
```

```
  Logger.log(`Created Folder ID ${folder.getId()}`);
}
```

The above code defines two functions that use the Google Drive API to create new folders in a Google Drive account.

The cFolder function creates a new folder in the root directory of the user's Google Drive account. It starts by setting the desired folder name in a variable called folderName. It then calls the DriveApp.createFolder(folderName) method to create a new folder with the specified name, and stores the resulting folder object in a variable called folder. Finally, it logs a message to the console indicating the ID of the new folder that was created using the Logger.log() method.

The addFolder function creates a new folder inside an existing folder with a specific ID. It starts by setting the desired folder name in a variable called folderName, and the ID of the parent folder where the new folder will be created in a variable called id. It then calls the DriveApp.getFolderById(id) method to retrieve the parent folder object based on the specified ID. It then calls the createFolder(folderName) method on the parent folder object to create a new folder with the specified name, and stores the resulting folder object in a variable called folder. Finally, it logs a message to the console indicating the ID of the new folder that was created using the Logger.log() method.

Overall, these functions demonstrate how to use the Google Drive API in Google Apps Script to create new folders in a Google Drive account. The first function creates a new folder in the root directory, while the second function creates a new folder inside an existing folder by specifying its parent folder ID. These functions

could be useful for automating file organization or for creating new folders as part of a larger Google Apps Script workflow.

Send an email when a Google Form is submitted:

```
function sendEmail(e) {
  var name = e.namedValues["Name"][0];
  var email = e.namedValues["Email"][0];
  var message = e.namedValues["Message"][0];
  var subject = "New message from " + name;
  var body = "Name: " + name + "\nEmail: " + email +
"\n\nMessage:\n" + message;
  MailApp.sendEmail("youremail@example.com", subject,
body);
}
```

This script sends an email to "youremail@example.com" when someone submits a Google Form. It extracts the form data using the namedValues property of the form submission event object.

Use form data in Email

https://youtu.be/D3cxVxWrzWU

```
function onFormSubmit(e){
  const vals = e.namedValues;
  const subject = 'New Form Submission';
  const email = 'gappscourses@gmail.com';
  const message = JSON.stringify(vals);
  const body = `Message ${vals.message[0]} Name
${vals.name[0]}  Name ${vals["Email Address"][0]}`;
```

```
  MailApp.sendEmail(email,subject,body);
}
```

Set up a trigger to capture the form submission.

Generate a PDF from a Google Doc and save it to Google Drive:

```
function generatePDF() {
  var docFile = DriveApp.getFileById("DOCUMENT_ID");
  var docName = docFile.getName();
  var pdfFile =
DriveApp.createFile(docFile.getAs("application/pdf"));
  pdfFile.setName(docName + ".pdf");
}
```

This script generates a PDF file from a Google Doc with the ID "DOCUMENT_ID" and saves it to the root folder of Google Drive. It uses the getAs method of the file object to convert the Doc to a PDF.

Create a custom menu in Google Sheets:

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu("My Menu")
    .addItem("Item 1", "menuItem1")
    .addItem("Item 2", "menuItem2")
    .addToUi();
}
```

```
function menuItem1() {
  Browser.msgBox("You clicked Item 1!");
}

function menuItem2() {
  Browser.msgBox("You clicked Item 2!");
}
```

This script creates a custom menu in a Google Sheets document. When the user clicks on one of the menu items, it displays a message box with a corresponding message. The onOpen function is a special function that runs automatically when the sheet is opened.

Import data from a CSV file and write it to a Google Sheet:

```
function importCSV() {
  var file =
DriveApp.getFilesByName("FILE_NAME").next();
  var csvData =
Utilities.parseCsv(file.getBlob().getDataAsString());
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.getRange(1, 1, csvData.length,
csvData[0].length).setValues(csvData);
}
```

This script imports data from a CSV file with the name "FILE_NAME" and writes it to the active sheet in the current Google Sheets document. It uses the parseCsv method of the Utilities class to parse the CSV data into a 2D array, and the setValues method of the sheet object to write the data to the sheet.

Add a trigger to run a function on a schedule:

```
function myFunction() {
  Logger.log("This function runs on a schedule!");
}

function createTrigger() {
  ScriptApp.newTrigger("myFunction")
    .timeBased()
    .everyHours(1)
    .create();
}
```

This script creates a trigger that runs the myFunction function every hour. The createTrigger function uses the newTrigger method of the ScriptApp class to create the trigger, and the timeBased method to specify that it should run on a schedule.

Send email from a Google Sheet:

```
function sendEmail() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var emailRange = sheet.getRange("A2:B5");
  var emailValues = emailRange.getValues();

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

```
    var subject = emailValues[i][1];
    var body = "Dear " + recipient + ",\n\nThis is an
automated email.\n\nBest regards,\nYour Name";
    MailApp.sendEmail(recipient, subject, body);
  }
}
```

This script sends an automated email to a list of recipients and subjects specified in cells A2:B5 of the active sheet. The recipient and subject are read from the sheet, and the body of the email is generated dynamically.

Create a custom menu in Google Sheets:

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom Menu')
    .addItem('Sort A-Z', 'sortAtoZ')
    .addItem('Sort Z-A', 'sortZtoA')
    .addToUi();
}

function sortAtoZ() {
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.getRange("A2:C10").sort(1);
}

function sortZtoA() {
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.getRange("A2:C10").sort(1, false);
}
```

This script creates a custom menu in Google Sheets with two options to sort data in ascending or descending order. When the user selects one of the options, the corresponding function is called to sort the data in the active sheet.

Automate Google Forms:

```
function onFormSubmit(e) {
  var sheet = SpreadsheetApp.getActiveSheet();
  var range = sheet.getRange(sheet.getLastRow(), 1, 1,
sheet.getLastColumn());
  var values = range.getValues()[0];

  var recipient = values[0];
  var subject = "New Form Submission";
  var body = "Hello,\n\nA new form has been
submitted.\n\nThank you.";
  MailApp.sendEmail(recipient, subject, body);
}
```

This script sends an email notification to the recipient specified in the first column of the sheet when a new form submission is received. The function is triggered automatically when a new form submission is added to the sheet.

Create a custom Google Drive folder:

```
function createFolder() {
  var folderName = "My Custom Folder";
  var parentFolder = DriveApp.getRootFolder();
```

```
  var newFolder =
parentFolder.createFolder(folderName);
  Logger.log("New folder created: " +
newFolder.getName());
}
```

This script creates a new folder called "My Custom Folder" in the root folder of Google Drive. The script logs the name of the newly created folder for verification.

Generate a random password:

```
function generatePassword() {
  var length = 12;
  var charset =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
23456789";
  var password = "";

  for (var i = 0; i < length; i++) {
    password += charset.charAt(Math.floor(Math.random()
* charset.length));
  }

  Logger.log("New password generated: " + password);
}
```

This script generates a random password with a length of 12 characters using a combination of lowercase and uppercase letters and numbers. The script logs the newly generated password for verification.

Create a custom function in Google Sheets:

```
function addNumbers(a, b) {
  return a + b;
}
```

This script creates a custom function called addNumbers that takes two arguments and returns their sum. To use the custom function in a cell in Google Sheets, type =addNumbers(a, b) where a and b are the values you want to add.

Copy a sheet to a new spreadsheet:

```
function copySheet() {
  var sheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName("S
heet1");
  var newSpreadsheet = SpreadsheetApp.create("New
Spreadsheet");
  var destinationSheet =
newSpreadsheet.getActiveSheet();

  sheet.copyTo(destinationSheet);

newSpreadsheet.deleteSheet(newSpreadsheet.getSheetByNam
e("Sheet1"));
  Logger.log("Sheet copied to new spreadsheet.");
}
```

This script copies the contents of a sheet called "Sheet1" in the active spreadsheet to a new spreadsheet called "New

Spreadsheet". The script then deletes the default sheet in the new spreadsheet and logs a message for verification.

Extract data from a PDF file:

```
function extractPDF() {
  var file = DriveApp.getFilesByName("My PDF
File.pdf").next();
  var blob = file.getBlob();
  var data = extractTextFromPDF(blob);

  Logger.log(data);
}

function extractTextFromPDF(blob) {
  var resource = {
    title: blob.getName(),
    mimeType: blob.getContentType()
  };
  var options = {
    ocr: true,
    ocrLanguage: "en"
  };
  var text = Drive.Files.insert(resource, blob,
options).ocrResult.text;

  return text;
}
```

This script extracts text from a PDF file called "My PDF File.pdf" stored in Google Drive using OCR (Optical Character Recognition).

The script first retrieves the file as a blob and then passes it to a helper function called extractTextFromPDF. The function uses the Drive API to insert the blob as a new file with OCR enabled and returns the extracted text.

Merge cells in Google Sheets:

```
function mergeCells() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var range = sheet.getRange("A1:B2");

  range.merge();
  Logger.log("Cells merged.");
}
```

This script merges the cells in the range A1:B2 in the active sheet in Google Sheets. The script logs a message for verification.

Export data from Google Sheets to a CSV file:

```
function exportCSV() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var range = sheet.getDataRange();
  var values = range.getValues();
  var csv = "";

  for (var i = 0; i < values.length; i++) {
    csv += values[i].join(",") + "\n";
  }
```

```
  var file = DriveApp.createFile("Exported Data.csv",
csv, MimeType.CSV);
  Logger.log("CSV file created: " + file.getName());
}
```

This script exports the data from the active sheet in Google Sheets to a CSV file called "Exported Data.csv" stored in Google Drive. The script reads the values in the sheet and converts them to a CSV format. The script then creates a new file in Google Drive with the exported data and logs a message for verification.