Apps Script Coding Examples

Creating a Google Calendar Event

This example shows how to create a new event on a Google Calendar using Google Apps Script.

```
function createCalendarEvent() {
  var calendarId = "your_calendar_id";
  var calendar =
CalendarApp.getCalendarById(calendarId);
  var event = calendar.createEvent('Test Event', new
Date('March 10, 2023 09:00:00'), new Date('March 10,
2023 10:00:00'));
}
```

Explanation:

In this script, we first define the calendar ID of the Google Calendar we want to create the event on. We then call the CalendarApp.getCalendarById() function to get the calendar with the specified ID. We then create a new event on the calendar

using the createEvent() function and passing the event title, start date, and end date as arguments.

Extracting Data from a Google Form

This example shows how to extract data from a Google Form using Google Apps Script.

```
function extractFormData() {
  var formId = "your_form_id";
  var form = FormApp.openById(formId);
  var responses = form.getResponses();
  for (var i = 0; i < responses.length; i++) {
    var response = responses[i];
    var itemResponses = response.getItemResponses();
    for (var j = 0; j < itemResponses.length; j++) {
      var itemResponse = itemResponses[j];
      var question = itemResponse.getItem().getTitle();
      var answer = itemResponse.getResponse();
      Logger.log("Question: " + question + ", Answer: "
+ answer);
    }
  }
}
```

Explanation:

In this script, we first define the form ID of the Google Form we want to extract data from. We then call the FormApp.openById() function to open the form with the specified ID. We then get all the form responses using the getResponses() function and loop through each response. For each response, we get the item responses using the getItemResponses() function and loop through each item response. We then get the question and answer for each item response using the getTitle() and getResponse() functions and log them to the console.

Creating a Google Slides Presentation

This example shows how to create a new presentation on Google Slides using Google Apps Script.

```
function createPresentation() {
  var presentationTitle = "Test Presentation";
  var presentation =
SlidesApp.create(presentationTitle);
  var slide = presentation.getSlides()[0];
  slide.insertShape(SlidesApp.ShapeType.RECTANGLE, 0,
0, 100, 100);
}
```

Explanation:

In this script, we first define the title of the new presentation we want to create. We then call the SlidesApp.create() function to

create a new presentation with the specified title. We then get the first slide of the presentation using the getSlides() function and insert a rectangle shape on the slide using the insertShape() function.

Using Google Translate API

This example shows how to use the Google Translate API to translate text using Google Apps Script.

```
function translateText() {
  var textToTranslate = "Hello World!";
  var targetLanguage = "es";
  var translatedText =
LanguageApp.translate(textToTranslate, '',
targetLanguage);
  Logger.log("Translated Text: " + translatedText);
}
```

Explanation:

In this script, we first define the text we want to translate and the target language we want to translate the text into. We then call the LanguageApp.translate() function to translate the text using the Google Translate API and passing the text, source language, and target language as arguments. We then log the translated text to the console using the Logger.log() function.

Copying a Google Drive Folder

This example shows how to copy a Google Drive folder using Google Apps Script.

```
function copyFolder() {
  var folderId = "your_folder_id";
  var folder = DriveApp.getFolderById(folderId);
  var newFolderName = folder.getName() + " - Copy";
  var newFolder = DriveApp.createFolder(newFolderName);
  var files = folder.getFiles();
  while (files.hasNext()) {
    var file = files.next();
    file.makeCopy(file.getName(), newFolder);
  }
}
```

Explanation:

In this script, we first define the folder ID of the Google Drive folder we want to copy. We then call the DriveApp.getFolderById() function to get the folder with the specified ID. We then create a new folder with the same name as the original folder but with " - Copy" appended to it using the DriveApp.createFolder() function. We then get all the files in the original folder using the getFiles() function and loop through each file. For each file, we make a copy of the file in the new folder using the makeCopy() function and passing the file name and the new folder as arguments.

Adding a Custom Menu to a Google Sheet

This example shows how to add a custom menu to a Google Sheet using Google Apps Script.

```
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom Menu')
      .addItem('First Item', 'menuItem1')
      .addSeparator()
      .addItem('Second Item', 'menuItem2')
      .addToUi();
}

function menuItem1() {
  // Do something for the first menu item
}

function menuItem2() {
  // Do something for the second menu item
}
```

Explanation:

In this script, we first define the onOpen() function, which is a special function that runs automatically when the Google Sheet is

opened. We then use the SpreadsheetApp.getUi() function to get the UI service for the Google Sheet. We then create a new custom menu using the createMenu() function and add two menu items using the addItem() function. We then define the two menu item functions menuItem1() and menuItem2() which will be executed when the corresponding menu item is clicked.

Converting a Google Sheet to PDF

This example shows how to convert a Google Sheet to PDF using Google Apps Script.

```
function convertToPDF() {
  var sheetId = "your_sheet_id";
  var sheet =
SpreadsheetApp.openById(sheetId).getSheetByName("Sheet1
");
  var blob =
sheet.getBlob().getAs("application/pdf").setName("Sheet
1.pdf");
  DriveApp.createFile(blob);
}
```

Explanation:

In this script, we first define the sheet ID of the Google Sheet we want to convert to PDF. We then call the

SpreadsheetApp.openById() function to open the sheet with the specified ID. We then get the sheet with the name "Sheet1" using the getSheetByName() function. We then get the sheet data as a PDF blob using the getBlob() and getAs() functions. We then create a new file in Google Drive using the DriveApp.createFile() function.

Setting a Google Sheet's Cell Value

This example shows how to set a cell value in a Google Sheet using Google Apps Script.

```
function setCellValue() {
  var sheetId = "your_sheet_id";
  var sheet =
SpreadsheetApp.openById(sheetId).getSheetByName("Sheet1
");
  sheet.getRange("A1").setValue("Hello World!");
}
```

Explanation:

In this script, we first define the sheet ID of the Google Sheet we want to set the cell value for. We then call the SpreadsheetApp.openById() function to open the sheet with the specified ID. We then get the sheet with the name "Sheet1" using the getSheetByName() function. We then set the value of cell A1 to "Hello World!" using the setValue() function.

Uploading a File to Google Drive

This example shows how to upload a file to Google Drive using Google Apps Script.

```
function uploadFile() {
  var file = DriveApp.createFile("Test File", "This is a test file");
  var folderId = "your_folder_id";
  var folder = DriveApp.getFolderById(folderId);
  folder.addFile(file);
}
```

Explanation:

In this script, we first create a new file in Google Drive using the DriveApp.createFile() function. We then define the folder ID of the Google Drive folder we want to upload the file to. We then get the folder with the specified ID using the DriveApp.getFolderById() function. We then add the newly created file to the folder using the addFile() function.

Sending an Email from Google Sheets

This example shows how to send an email from a Google Sheet using Google Apps Script.

```
function sendEmail() {
  var emailAddress = "recipient@example.com";
  var subject = "Test Email";
  var body = "This is a test email";
  MailApp.sendEmail(emailAddress, subject, body);
}
```

Explanation:

In this script, we first define the recipient email address, subject, and body of the email we want to send. We then call the MailApp.sendEmail() function to send the email to the specified recipient with the specified subject and body. Note that this function requires the necessary authorization to be granted in order to send emails.

Sending Emails

Google Apps Script can be used to send emails directly from Google Sheets, Docs, or Forms. This is useful when you need to send automated emails to a large number of recipients or when you need to customize the content of the email for each recipient. Here's an example code:

```
function sendEmail() {
  var recipient = "example@example.com";
```

```
  var subject = "Test Email";
  var body = "This is a test email.";
  MailApp.sendEmail(recipient, subject, body);
}
```

Creating a Calendar Event

Google Apps Script can also be used to create calendar events directly from Google Sheets or Forms. Here's an example code:

```
function createEvent() {
  var calendar = CalendarApp.getDefaultCalendar();
  var title = "Test Event";
  var start = new Date("2023-03-05T12:00:00Z");
  var end = new Date("2023-03-05T13:00:00Z");
  calendar.createEvent(title, start, end);
}
```

Creating a Google Form

Google Apps Script can be used to create Google Forms programmatically. This is useful when you need to create a large number of forms or when you need to customize the content of the form for each respondent. Here's an example code:

```
function createForm() {
  var form = FormApp.create("Test Form");
  form.addTextItem().setTitle("What is your name?");
```

```
  form.addMultipleChoiceItem()
    .setTitle("What is your favorite color?")
    .setChoices([
      form.createChoice("Red"),
      form.createChoice("Blue"),
      form.createChoice("Green"),
    ]);
}
```

Reading Data from Google Sheets

Google Apps Script can be used to read data from Google Sheets and process it programmatically. Here's an example code that reads the values from the first row of a sheet:

```
function readData() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var values = sheet.getRange("1:1").getValues()[0];
  Logger.log(values);
}
```

Writing Data to Google Sheets

Google Apps Script can also be used to write data to Google Sheets. Here's an example code that writes the values "John" and "Doe" to the first row of a sheet:

```
function writeData() {
```

```
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.getRange("A1:B1").setValues([["John", "Doe"]]);
}
```

Send Emails Automatically:

With Apps Script, you can write a script that sends an email automatically to a recipient or a group of recipients at a specific time or interval. The code below sends an email message containing a subject and a body to a specified email address.

```
function sendEmail() {
  var recipient = "recipient@email.com";
  var subject = "Hello from Apps Script";
  var body = "This email was sent using Google Apps Script!";
  MailApp.sendEmail(recipient, subject, body);
}
```

Custom Emails and HTML option

```
function senderEmail(){
  const email = Session.getActiveUser().getEmail();
```

```
  const emailSend = 'gappscourses@gmail.com';
  const subject = 'Hello World';
  const body = '<h1>Laurence Svekis</h1>';
  MailApp.sendEmail({
    to:email,
    subject:subject,
    htmlBody:body});
  Logger.log(email);
}
```

const email = Session.getActiveUser().getEmail(); - This line gets the email address of the currently active user using the Session.getActiveUser() method and stores it in the email constant. The getEmail() method retrieves the email address associated with the user's account.

const emailSend = 'gappscourses@gmail.com'; - This line defines a constant variable named emailSend that contains the email address of the recipient.

const subject = 'Hello World'; - This line defines a constant variable named subject that contains the subject of the email message.

const body = '<h1>Laurence Svekis</h1>'; - This line defines a constant variable named body that contains the body of the email message. It is in HTML format.

MailApp.sendEmail({to:email, subject:subject, htmlBody:body}); - This line uses the MailApp service to send an email message to the email address stored in the email constant. The email message contains the subject stored in the subject constant and the body stored in the body constant. The htmlBody option specifies that the body of the message is in HTML format.

Logger.log(email); - This line logs the email address of the currently active user to the execution log using the Logger.log() method.

Create Google Forms:

Google Forms is a powerful tool for collecting data from users. With Apps Script, you can programmatically create Google Forms, customize them, and collect the responses. The code below creates a new Google Form and adds a multiple-choice question with two options.

```
function createForm() {
  var form = FormApp.create("My Form");
```

```
  var item = form.addMultipleChoiceItem();
  item.setTitle("Choose an option")
      .setChoices([
          item.createChoice("Option 1"),
          item.createChoice("Option 2")
      ]);
}
```

Creating a custom form

```
function creatorForm(){
  const form = FormApp.create('Laurence Svekis');
  const item = form.addMultipleChoiceItem();
  item.setTitle('Select One')
    .setChoices([
      item.createChoice('One'),
      item.createChoice('Two'),
      item.createChoice('Three')
      ]);
}
```

The creatorForm() function creates a Google Form and adds a multiple choice question to it. Here is a line-by-line explanation of the code:

const form = FormApp.create('Laurence Svekis'); - This line creates a new Google Form with the title "Laurence Svekis" using the FormApp.create() method. The create() method returns a Form object that represents the newly created form.

const item = form.addMultipleChoiceItem(); - This line adds a multiple choice question to the form using the addMultipleChoiceItem() method of the Form object. The addMultipleChoiceItem() method returns a MultipleChoiceItem object that represents the newly created question.

item.setTitle('Select One') - This line sets the title of the multiple choice question to "Select One" using the setTitle() method of the MultipleChoiceItem object.

.setChoices([- This line begins an array of answer choices for the multiple choice question using the setChoices() method of the MultipleChoiceItem object. The setChoices() method takes an array of Choice objects as its argument.

item.createChoice('One'), - This line creates a new answer choice with the text "One" using the createChoice() method of the MultipleChoiceItem object. The createChoice() method returns a Choice object that represents the newly created answer choice.

item.createChoice('Two'), - This line creates a new answer choice with the text "Two" using the createChoice() method of the MultipleChoiceItem object.

item.createChoice('Three') - This line creates a new answer choice with the text "Three" using the createChoice() method of the MultipleChoiceItem object.

]); - This line ends the array of answer choices for the multiple choice question using the setChoices() method of the MultipleChoiceItem object.

After executing the creatorForm() function, a new Google Form titled "Laurence Svekis" will be created with a multiple choice question that has three answer choices: "One", "Two", and "Three".

Generate Google Docs:

Google Docs is a cloud-based word processing tool that allows users to create and edit documents. With Apps Script, you can programmatically generate Google Docs, populate them with data, and format them. The code below creates a new Google Doc and adds a paragraph of text.

```
function createDoc() {
  var doc = DocumentApp.create("My Doc");
  var body = doc.getBody();
  body.insertParagraph(0, "Hello from Apps Script!");
}
```

Create and update a Google Doc

```
function createmyDoc(){
  const doc = DocumentApp.create('Laurence Svekis');
  const body = doc.getBody();
  body.insertParagraph(0,'My new Doc, Laurence
Svekis');
}

function updatemyDoc(){
  const id = '1JJyHruUbZQv4Q5zLHTVebDDbs';
  const doc = DocumentApp.openById(id);
  const body = doc.getBody();
  body.insertParagraph(10,'*****Laurence Svekis');
}
```

The createmyDoc() function and the updatemyDoc() function both manipulate a Google Docs document. Here's what each function does:

createmyDoc(): This function creates a new Google Docs document titled "Laurence Svekis", inserts a paragraph into the document, and sets the text of the paragraph to "My new Doc, Laurence Svekis". Here's what each line of the function does:

- const doc = DocumentApp.create('Laurence Svekis');: This line creates a new Google Docs document titled "Laurence Svekis" using the create() method of the DocumentApp class. The create() method returns a Document object that represents the newly created document.
- const body = doc.getBody();: This line gets the Body object of the newly created document using the getBody() method of the Document object.
- body.insertParagraph(0,'My new Doc, Laurence Svekis');: This line inserts a new paragraph at the beginning of the document using the insertParagraph() method of the Body object. The insertParagraph() method takes two arguments: the position where the new paragraph should be inserted (in this case, the beginning of the document), and the text of the paragraph ("My new Doc, Laurence Svekis" in this case).

updatemyDoc(): This function updates an existing Google Docs document by inserting a new paragraph into it. Here's what each line of the function does:

- const id = '1JJyHruUbZQv5LHTVebDDbs';: This line sets the id variable to the ID of the Google Docs document that we want to update. This ID is a unique identifier that is assigned to each Google Docs document.
- const doc = DocumentApp.openById(id);: This line opens the Google Docs document with the specified ID using the openById() method of the DocumentApp class. The openById() method returns a Document object that represents the opened document.
- const body = doc.getBody();: This line gets the Body object of the opened document using the getBody() method of the Document object.
- body.insertParagraph(10,'*****Laurence Svekis');: This line inserts a new paragraph at position 10 (i.e., after the tenth paragraph) in the document using the insertParagraph() method of the Body object. The text of the new paragraph is "*****Laurence Svekis". The insertParagraph() method takes two arguments: the position where the new paragraph should be inserted (in this case, after the tenth paragraph), and the text of the paragraph ("*****Laurence Svekis" in this case).

Add Data to Google Sheets:

Google Sheets is a cloud-based spreadsheet tool that allows users to create and edit spreadsheets. With Apps Script, you can programmatically add data to a Google Sheet, retrieve data from it, and perform various operations. The code below adds a new row of data to a Google Sheet.

```
function addData() {
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.appendRow(["John Doe", "johndoe@email.com",
"555-1234"]);
}
```

Create and update a Google Sheet

```
function makeSheets(){
  const ss = SpreadsheetApp.create('Laurence Sheet');
  const sheet = ss.getSheets()[0];
  const row = ['Laurence','Svekis','100'];
  sheet.appendRow(row);
  Logger.log(ss.getId());
}
```

```
function updateSheet(){
  const id =
'1P9R_b-dTdoBAAWAGF6kPZyKSXJ6r2P_LFgrKeReNmrY';
  const ss = SpreadsheetApp.openById(id);
  const sheet = ss.getSheets()[0];
  const row = ['1 Laurence','1 Svekis','55'];
  sheet.appendRow(row);
}
```

The first function makeSheets() creates a new Google Sheets file with the name "Laurence Sheet" and then retrieves the first sheet in the file. It then creates a new row containing three values, 'Laurence', 'Svekis', and '100', and appends the row to the end of the sheet. Finally, it logs the ID of the created spreadsheet using the Logger class.

The second function updateSheet() opens an existing Google Sheets file using its ID and retrieves the first sheet in the file. It then creates a new row containing three values, '1 Laurence', '1 Svekis', and '55', and appends the row to the end of the sheet. The function updates the sheet in the file without creating a new sheet.

Automate Google Calendar:

Google Calendar is a powerful tool for managing events and schedules. With Apps Script, you can automate various tasks in Google Calendar, such as creating events, updating events, and sending notifications. The code below creates a new event in Google Calendar.

```
function createEvent() {
  var calendar = CalendarApp.getDefaultCalendar();
  var title = "My Event";
  var start = new Date("March 1, 2023 10:00:00");
  var end = new Date("March 1, 2023 11:00:00");
  var event = calendar.createEvent(title, start, end);
}
```

Add new event to Default Calendar

```
function myEventCal(){
  const cal = CalendarApp.getDefaultCalendar();
  const title = 'My Birthday';
  const start = new Date('March 1, 2023 10:00:00');
  const end = new Date('March 1, 2023 11:00:00');
  const event = cal.createEvent(title,start,end);
```

```
}
```

The function myEventCal() creates a new calendar event on the default calendar of the user. First, it retrieves the default calendar object using the getDefaultCalendar() method. Then, it creates variables for the title of the event, the start time, and the end time. In this case, the title of the event is "My Birthday", the start time is set for March 1, 2023, at 10:00 AM, and the end time is set for March 1, 2023, at 11:00 AM.

Finally, the createEvent() method is called on the cal object, passing in the title, start, and end variables to create a new event on the default calendar. The newly created event will be visible on the user's calendar at the specified start and end times with the specified title.