

# Google Apps Script

## Google Apps Script Tutorial: Automated Email Notifications on Google Sheets Edits

### **Description:**

In this Google Apps Script tutorial, we'll explore how to automate email notifications when a specific cell in a Google Sheets document is edited. This can be a handy tool for staying informed about important changes in your spreadsheet data.

The script utilizes the `onEdit` trigger, which is an event-driven function that runs whenever an edit occurs in the connected Google Sheets document.

Here's a breakdown of what the code does:

It first identifies the source spreadsheet and the active sheet where the edit takes place.

It specifies a target cell to monitor, in this case, 'A1'.

When a change is made in the specified cell ('A1'), the script triggers an email notification.


The email contains details such as the edited cell's range, the name of the sheet, and the new value that was entered.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

The email is sent to the specified recipient, in this case, 'gappscourses@gmail.com'.

By customizing the code to your specific use case and recipient email, you can implement automated notifications for any edits made in your Google Sheets document. This can be particularly useful for collaboration and data tracking.

Here's how you can set up this project:

1. Open your Google Sheets document.
2. Click on Extensions in the menu bar and then select Apps Script.
3. Delete any code that is already in the script editor.
4. Copy and paste the above code into the script editor.
5. Save your project with a name of your choice.
6. Run the onEdit function by clicking the play button .
7. Make sure that you grant the necessary permissions for the script to send emails and access the Google Sheets document.

Remember to adapt the code to your specific use case and requirements. Google Apps Script can be used for a wide range of tasks, from automating Google Sheets to creating custom add-ons for various Google Workspace apps.

```
function onEdit(e) {  
  const ss = e.source;  
  const sheet = ss.getActiveSheet();  
  const sName = sheet.getName();
```

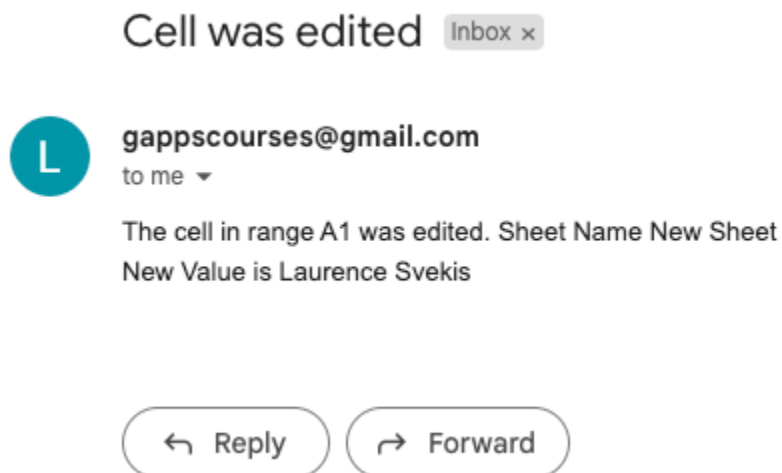
Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

```

const range = 'A1';
if(e.range.getA1Notation()===range){
  const subject = 'Cell was edited';
  const email = 'gappscourses@gmail.com';
  const message = `The cell in range ${range} was
edited. Sheet Name ${sName} \nNew Value is ${e.value}`;
  MailApp.sendEmail(email,subject,message);
}
}

```

	A	B
1	Laurence Svekis	
2	Svekis	



The provided code is a Google Apps Script function named onEdit. It is a simple event-triggered function that runs automatically whenever an edit is made in a Google Sheets document. Here's an explanation of what the code does:

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

1. `onEdit(e)`: This is a special function in Google Apps Script, which is automatically executed whenever an edit is made in the connected Google Sheets document. The `e` parameter is an event object that contains information about the edit event, such as the edited range, the new value, and the source spreadsheet.
2. `const ss = e.source;`: This line gets the source (the Google Sheets document) where the edit occurred and stores it in the `ss` variable.
3. `const sheet = ss.getActiveSheet();`: It retrieves the currently active sheet in the spreadsheet and stores it in the `sheet` variable.
4. `const sName = sheet.getName();`: This line fetches the name of the active sheet and stores it in the `sName` variable.
5. `const range = 'A1';`: It defines the cell range to monitor for edits, in this case, cell A1.
6. `if (e.range.getA1Notation() == range) {`: This conditional statement checks if the edited cell's A1 notation matches the range you want to monitor (A1).
7. `const subject = 'Cell was edited';`: It sets the email subject to "Cell was edited."
8. `const email = 'gappscourses@gmail.com';`: Specifies the recipient's email address. You can replace this with the actual email address you want to send the notification to.
9. `const message = The cell in range ${range} was edited. Sheet Name ${sName} \nNew Value is ${e.value};`: This constructs the email message. It includes the range that was edited (A1), the name of the sheet where the edit occurred, and the new value that was entered.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

10. `MailApp.sendEmail(email, subject, message);`: Finally, this line sends an email to the specified recipient (defined in the email variable) with the subject and message content you've set. The email serves as a notification that a cell within the specified range has been edited.

In summary, this Google Apps Script code is designed to send an email notification to the specified recipient whenever a change is made to cell A1 in a Google Sheets document. It provides information about the edited cell's range, the sheet's name, and the new value that was entered during the edit.

#### Code

```
function onEdit(e){
  const ss = e.source;
  const sheet = ss.getActiveSheet();
  const sName = sheet.getName();
  const range = 'A1';
  if(e.range.getA1Notation()==range){
    const subject = 'Cell was edited';
    const email = 'gappscourses@gmail.com';
    const message = `The cell in range ${range} was edited. Sheet
Name ${sName} \nNew Value is ${e.value}`;
    MailApp.sendEmail(email,subject,message);
  }
}
```

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

# Google Apps Script Tutorial: Creating a Custom Menu for Adding Current Date in Google Sheets

Description:

Welcome to this Google Apps Script tutorial! In this video, we'll walk you through the process of creating a custom menu in Google Sheets and using it to add the current date to any cell with just one click.

The code we'll be working with includes two functions:

`onOpen()`: This function is triggered when you open a Google Sheets document. It creates a custom menu labeled 'Custom Menu' in the Google Sheets user interface.

`addDate()`: This function is executed when you select the 'Current Date' option from the custom menu. It automates the process of inserting the current date into the selected cell.

We'll guide you step by step, explaining the code and how it works, so you can implement this automation in your own Google Sheets documents. Whether you're tracking deadlines, managing schedules, or simply need a quick way to timestamp your data, this tutorial has you covered. Stay tuned, and let's get started with Google Apps Script!

```
function onOpen() {
```

```
  const ui = SpreadsheetApp.getUi();
```

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

```

ui.createMenu('Custom Menu')
  .addItem('Current Date', 'adder')
  .addToUi();
}

function adder(){
  //SpreadsheetApp.getUi().alert('You clicked it');
  const sheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  const range = sheet.getActiveRange();
  const currentDate = new Date();
  range.setValue(currentDate);
}

```

**Here's an explanation of this script:**

1. `onOpen()`: This function is a built-in trigger function that runs when the Google Sheets document is opened. It creates a custom menu in the Google Sheets interface. The `ui.createMenu` method allows you to create a custom menu with a specific name ('Custom Menu') and add menu items to it.
2. `addItem('Insert Current Date and Time', 'insertCurrentDateAndTime')`: This line adds a menu item named 'Insert Current Date and Time' to the custom menu. When this menu item is clicked, it will execute the `insertCurrentDateAndTime` function.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

3. `addToUi()`: This attaches the custom menu to the Google Sheets interface.
4. `insertCurrentDateAndTime()`: This is the function that gets executed when the 'Insert Current Date and Time' menu item is clicked. It first identifies the active sheet and the currently selected range. Then, it gets the current date and time and inserts it into the selected cell.

### **To use this script:**

1. Open a Google Sheets document.
2. Click on Extensions in the menu bar and then select Apps Script.
3. Delete any code that is already in the script editor.
4. Copy and paste the provided code into the script editor.
5. Save your project with a name of your choice.
6. Close the script editor.

You'll now have a 'Custom Menu' in your Google Sheets document with an 'Insert Current Date and Time' option that you can use to add the current date and time to any selected cell.

The provided code is a Google Apps Script for Google Sheets. It creates a custom menu in Google Sheets, and when a menu item is selected, it triggers a function to add the current date and time to a selected cell. Here's a detailed explanation of the code:

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>



1. `function onOpen() {` This function is a built-in trigger function in Google Apps Script. It is automatically executed when the Google Sheets document is opened.
2. `const ui = SpreadsheetApp.getUi();` This line creates a variable `ui` that is set to the `Ui` service provided by `SpreadsheetApp`. The `Ui` service allows you to interact with the user interface of Google Sheets.
3. `ui.createMenu('Custom Menu')` The `ui.createMenu` method is used to create a custom menu in Google Sheets. In this case, it creates a menu named `'Custom Menu'`.
4. `.addItem('Current Date', 'adder')` The `.addItem` method adds a menu item to the custom menu. In this case, it adds an item with the label `'Current Date'`, and it specifies that the function `'adder'` should be executed when this menu item is clicked. The `'adder'` function is defined later in the code.
5. `.addToUi();` The `.addToUi` method attaches the custom menu to the Google Sheets user interface. When you open the Google Sheets document, you will see the `'Custom Menu'` in the menu bar.
6. `function adder() {` This is a user-defined function named `adder`. It gets executed when the `'Current Date'` menu item is selected.
7. `//SpreadsheetApp.getUi().alert('You clicked it');` This line is commented out (using `//`) and not active in the code. It's a placeholder for displaying an alert message when the menu item is clicked. You can uncomment it if you want to display an alert message.
8. `const sheet =`  
`SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();` This line retrieves the currently active spreadsheet using

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

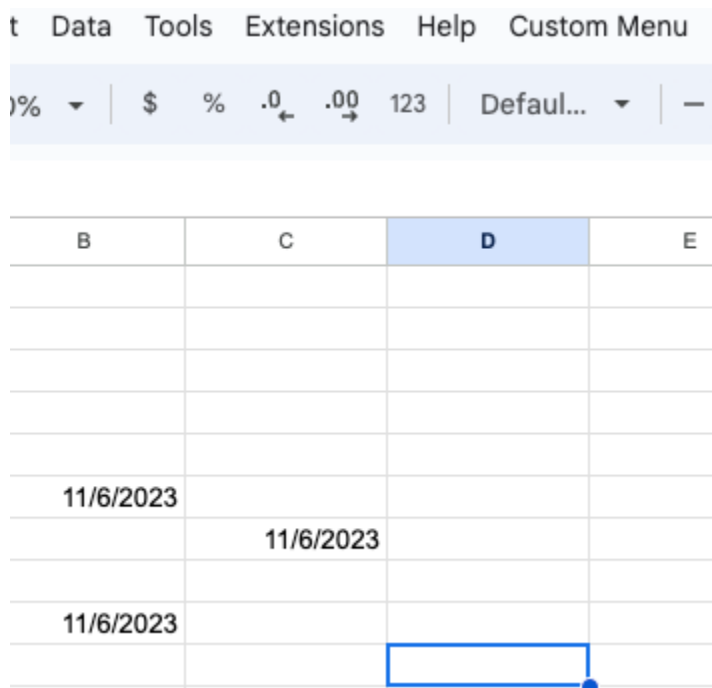
SpreadsheetApp.getActiveSpreadsheet(), and then it gets the active sheet within that spreadsheet using .getActiveSheet(). The sheet variable now represents the currently active sheet.

9. `const range = sheet.getActiveRange();` This line gets the currently selected range within the active sheet and stores it in the range variable.

10. `const currentDate = new Date();` It creates a JavaScript Date object, representing the current date and time.

11. `range.setValue(currentDate);` Finally, this line sets the value of the selected cell range (as determined by range) to the current date and time, effectively adding the current date and time to the selected cell.

### To use this script:



Open a Google Sheets document.

1. Click on Extensions in the menu bar and then select Apps Script.
2. Delete any code that is already in the script editor.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

3. Copy and paste the provided code into the script editor.
4. Save your project with a name of your choice.
5. Close the script editor.

You will now have a 'Custom Menu' in your Google Sheets document with an 'Current Date' option that you can use to add the current date and time to any selected cell.

## Google Apps Script Tutorial: Create a Custom Menu to Clear Cells in Google Sheets

Description:

Welcome to our Google Apps Script tutorial! In this video, we'll show you how to streamline your Google Sheets workflow by creating a custom menu that allows you to clear the content of selected cells with a single click.

We'll walk you through the provided code, which consists of two functions:

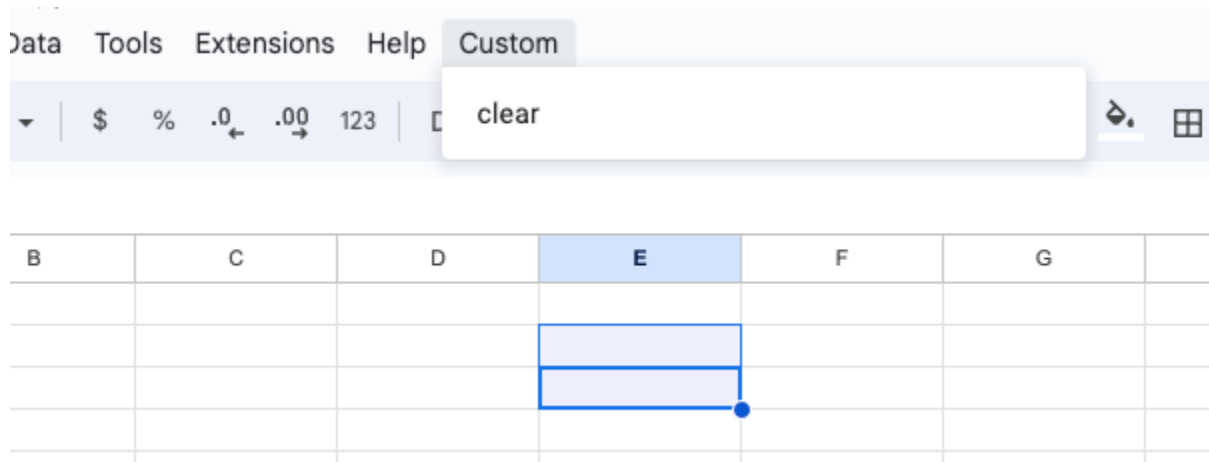
`onOpen()`: This function is triggered when you open a Google Sheets document. It sets up a custom menu labeled 'Custom' in the Google Sheets user interface.

`delCells()`: This function is executed when you select the 'clear' option from the custom menu. It automates the process of clearing the content from the currently selected cells.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

We'll provide a detailed explanation of the code and demonstrate how to implement it in your own Google Sheets. Whether you need to quickly erase data, reset values, or enhance your data management, this tutorial will help you boost your productivity. Stay tuned and let's dive into the world of Google Apps Script!

### Clear Selected Cells in Sheets



```
function onOpen(){
  const ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom')
    .addItem('clear','delCells')
    .addToUi();
}
```

```
function delCells(){
  const sheet =
  SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  const range = sheet.getActiveRange();
```

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

```
range.setValue('');  
}
```

1. **function onOpen()** { This is a user-defined function named onOpen. This function is a built-in trigger function in Google Apps Script and is automatically executed when the Google Sheets document is opened.
2. **const ui = SpreadsheetApp.getUi();** This line creates a variable ui and assigns it the Ui service provided by SpreadsheetApp. The Ui service allows you to interact with the user interface of Google Sheets.
3. **ui.createMenu('Custom')** The ui.createMenu method is used to create a custom menu in Google Sheets. In this case, it creates a menu with the label 'Custom'.
4. **.addItem('clear', 'delCells')** The .addItem method adds a menu item to the custom menu. In this case, it adds an item with the label 'clear'. When this menu item is clicked, it specifies that the function 'delCells' should be executed. The 'delCells' function is defined later in the code.
5. **.addToUi();** The .addToUi method attaches the custom menu to the Google Sheets user interface. When you open the Google Sheets document, you will see the 'Custom' menu in the menu bar.
6. **function delCells()** {This is a user-defined function named delCells. It gets executed when the 'clear' menu item is selected.
7. **const sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();**  
This line retrieves the currently active spreadsheet using SpreadsheetApp.getActiveSpreadsheet(), and then it gets the active sheet

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

within that spreadsheet using `.getActiveSheet()`. The sheet variable now represents the currently active sheet.

8. **`const range = sheet.getActiveRange();`** This line gets the currently selected range within the active sheet and stores it in the range variable.
9. **`range.setValue("");`** Finally, this line sets the value of the selected cell range (as determined by range) to an empty string, effectively clearing the content of the selected cells.

To use this script:

1. Open a Google Sheets document.
2. Click on Extensions in the menu bar and then select Apps Script.
3. Delete any code that is already in the script editor.
4. Copy and paste the provided code into the script editor.
5. Save your project with a name of your choice.
6. Close the script editor.

You will now have a 'Custom' menu in your Google Sheets document with a 'clear' option that you can use to clear the content of any selected cells.

```
function onOpen() {  
  var ui = SpreadsheetApp.getUi();  
  ui.createMenu('Custom Menu')  
    .addItem('Format Cells', 'formatSelectedCells')  
    .addToUi();  
}
```

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

```

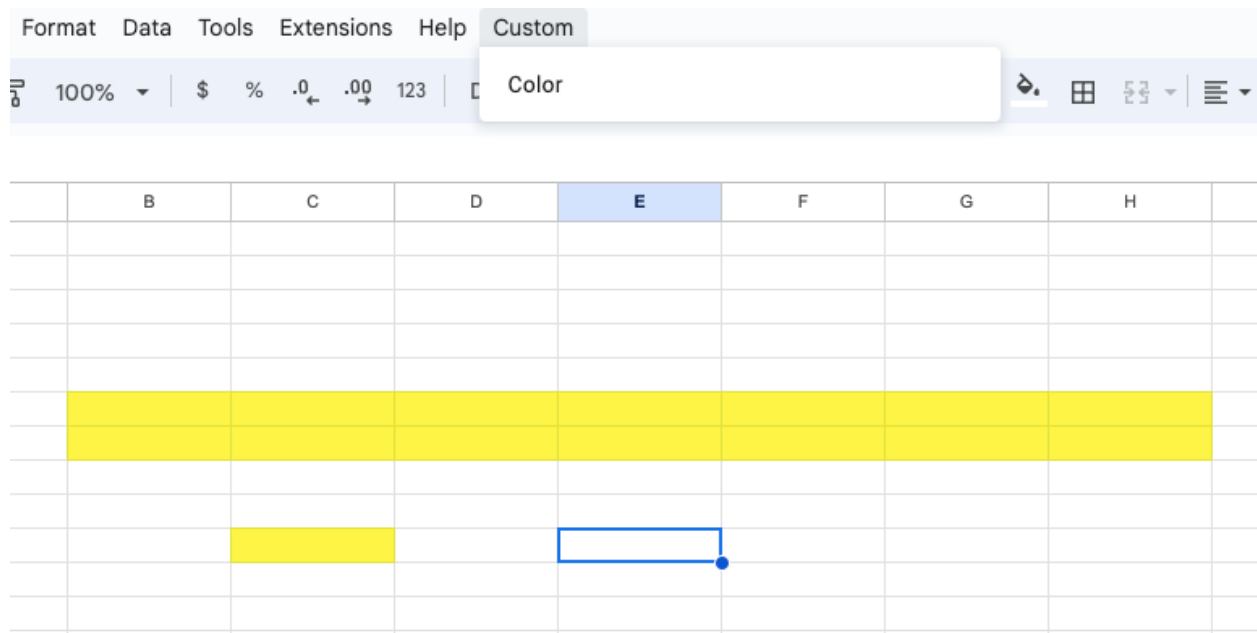
function formatSelectedCells() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  var range = sheet.getActiveRange();

  // Define the background color (e.g., light yellow)
  var color = "#FFFF99";

  // Apply the background color to the selected cells
  range.setBackground(color);
}

```

## Set Background Color



Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

```
function onOpen(){
  const ui = SpreadsheetApp.getUi();
  ui.createMenu('Custom')
  .addItem('Color','backGroundColor')
  .addToUi();
}
```

```
function backGroundColor(){
  //SpreadsheetApp.getUi().alert('clicked');
  const sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
  const range = sheet.getActiveRange();
  const color = '#FFFF00';
  range.setBackground(color);
}
```

Here's an explanation of this script:

1. `onOpen()`: This function is a built-in trigger function in Google Apps Script and runs automatically when the Google Sheets document is opened. It creates a custom menu in the Google Sheets interface.
2. `var ui = SpreadsheetApp.getUi();`: This line creates a variable `ui` and assigns it the `Ui` service provided by `SpreadsheetApp`. The `Ui` service allows you to interact with the user interface of Google Sheets.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>



3. `ui.createMenu('Custom Menu')`: The `ui.createMenu` method is used to create a custom menu in Google Sheets. In this case, it creates a menu with the label 'Custom Menu'.
4. `.addItem('Format Cells', 'formatSelectedCells')`: The `.addItem` method adds a menu item to the custom menu. It adds an item with the label 'Format Cells'. When this menu item is clicked, it specifies that the function 'formatSelectedCells' should be executed. The 'formatSelectedCells' function is defined later in the code.
5. `.addToUi()`: The `.addToUi` method attaches the custom menu to the Google Sheets user interface. When you open the Google Sheets document, you will see the 'Custom Menu' in the menu bar.
6. `formatSelectedCells()`: This is the function that gets executed when the 'Format Cells' menu item is clicked. It first identifies the active sheet and the currently selected range. Then, it defines a background color (in this case, light yellow) and applies that color to the selected cells using `range.setBackground(color)`.

To use this script:

1. Open a Google Sheets document.
2. Click on Extensions in the menu bar and then select Apps Script.
3. Delete any code that is already in the script editor.
4. Copy and paste the provided code into the script editor.
5. Save your project with a name of your choice.
6. Close the script editor.

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses <https://basescripts.com/>

You will now have a 'Custom Menu' in your Google Sheets document with a 'Format Cells' option that you can use to format the background color of any selected cells.