

Google Apps Script Examples of Custom Formulas in action in Sheets

CONVERT_TEMPERATURES Temperature Conversion in Google Apps Script - How to Convert Celsius to Fahrenheit and Vice Versa	1
GENERATE_PASSWORD Generate Secure Random Passwords with Google Apps Script	9
Optimize Cost Calculations with Google Apps Script: Learn How!	15
Unlock the Power of Weighted Averages in Google Sheets with Google Apps Script	25
CONVERT_TO_LETTER_GRADE	31
Custom formula to calculate average rating	38

CONVERT_TEMPERATURES Temperature Conversion in Google Apps Script - How to Convert Celsius to Fahrenheit and Vice Versa

<https://youtu.be/wwY45IJxfOc>

In this example, we'll create a custom formula that converts temperatures from Celsius to Fahrenheit and vice versa.

Scenario: You want to create a custom formula that converts temperatures between Celsius and Fahrenheit using the conversion formulas:

$F=9/5C+32$ for Celsius to Fahrenheit, and

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

$C=5/9(F-32)$ for Fahrenheit to Celsius.

B2 | fx =CONVERT_TEMPERATURES(A2:A4,"C","F")

	A	B	C
1	Temperature	Celsius	Fahrenheit
2	25	77	-3.888888889
3	70	158	21.11111111
4	0	32	-17.77777778

```
function CONVERT_TEMPERATURES(temperatures,fromUnit,toUnit){
const convertedTemps = [];
for(let i=0;i<temperatures.length;i++){
const temperature = temperatures[i];
let convertedTemperature = 0;
if(fromUnit.toLowerCase() === 'c' && toUnit.toLowerCase() ===
'f'){
convertedTemperature = (9/5)*temperature +32;
}else if(fromUnit.toLowerCase() === 'f' && toUnit.toLowerCase()
=== 'c'){
convertedTemperature = (5/9)*(temperature -32);
}else{
convertedTemperature = temperature;
}
convertedTemps.push([convertedTemperature]);
}
```

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

```
}  
return convertedTemps;  
}
```

This Google Apps Script function, named `CONVERT_TEMPERATURES`, is designed to convert a list of temperatures from one temperature unit to another. The function takes three parameters:

1. `temperatures`: An array of temperatures you want to convert.
2. `fromUnit`: A string representing the source temperature unit, which can be either 'C' (Celsius) or 'F' (Fahrenheit).
3. `toUnit`: A string representing the target temperature unit, which can also be either 'C' or 'F'.

Here's a step-by-step explanation of how the function works:

1. `const convertedTemps = []`: This initializes an empty array called `convertedTemps` to store the converted temperatures.
2. The function uses a for loop to iterate over each temperature in the `temperatures` array:

```
for (let i = 0; i < temperatures.length; i++) {  
  // ...  
}
```

3. Inside the loop, it retrieves the current temperature value and stores it in a variable `temperature`:

```
const temperature = temperatures[i];
```

4. It initializes a variable `convertedTemperature` to 0. This variable will store the converted temperature for the current value of `temperature`.

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

```
let convertedTemperature = 0;
```

5. The function checks the source and target temperature units using conditional statements. It uses `toLowerCase()` to make the comparison case-insensitive.
 - If `fromUnit` is 'C' (Celsius) and `toUnit` is 'F' (Fahrenheit), it converts the temperature from Celsius to Fahrenheit using the formula: $(9/5) * \text{temperature} + 32$.
 - If `fromUnit` is 'F' (Fahrenheit) and `toUnit` is 'C' (Celsius), it converts the temperature from Fahrenheit to Celsius using the formula: $(5/9) * (\text{temperature} - 32)$.
 - If the source and target units are the same or not recognized, it leaves the temperature unchanged.
6. The converted temperature is added to the `convertedTemps` array as a single-element array:

```
convertedTemps.push([convertedTemperature]);
```

7. After processing all temperatures in the `temperatures` array, the function returns the `convertedTemps` array, which contains the converted temperatures.

To use this function, you can call it with an array of temperatures, the source unit, and the target unit, and it will return an array of converted temperatures in the specified target unit. For example:

```
const temperatures = [0, 100, 25];
```

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

```
const convertedTemperatures =  
  CONVERT_TEMPERATURES(temperatures, 'C', 'F');
```

The convertedTemperatures variable will contain an array of temperatures converted from Celsius to Fahrenheit.

Data Table:

A	B	C	D
Temperature	Celsius	Fahrenheit	Converted
25			
70			
0			
Total			

Step 1: Setting up the Spreadsheet

Create a new Google Sheets document.

Enter your temperature values in column A starting from row 2.

Leave cells in columns B, C, and D empty for now.

In cell B1, enter "Celsius" as a header.

In cell C1, enter "Fahrenheit" as a header.

In cell D1, enter "Converted" as a header.

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

Step 2: Writing the Google Apps Script Code

Click on "Extensions" in the top menu, then select "Apps Script".

Delete any default code and replace it with the following script:

```
// Custom formula to convert temperatures between
Celsius and Fahrenheit
function CONVERT_TEMPERATURES(temperatures, fromUnit,
toUnit) {
  var convertedTemps = [];

  for (var i = 0; i < temperatures.length; i++) {
    var temperature = temperatures[i];
    var convertedTemperature = 0;

    if (fromUnit.toLowerCase() === "c" &&
toUnit.toLowerCase() === "f") {
      convertedTemperature = (9 / 5) * temperature +
32;
    } else if (fromUnit.toLowerCase() === "f" &&
toUnit.toLowerCase() === "c") {
      convertedTemperature = (5 / 9) * (temperature -
32);
    } else {
```

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

```

        convertedTemperature = temperature; // No
conversion needed
    }

    convertedTemps.push([convertedTemperature]);
}

return convertedTemps;
}

```

Step 3: Using the Custom Formula in Google Sheets

Go back to your Google Sheets document.

In cell B2, enter the following formula:

=CONVERT_TEMPERATURES(A2:A, "C", "F")

In cell C2, enter the following formula:

=CONVERT_TEMPERATURES(A2:A, "F", "C")

Explanation of the Code:

The function CONVERT_TEMPERATURES converts temperatures between Celsius and Fahrenheit (and vice versa).

It takes three parameters: temperatures, fromUnit, and toUnit.

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

- temperatures: The range of cells containing the temperature values.
- fromUnit: The unit of the temperature to convert from ("C" for Celsius, "F" for Fahrenheit).
- toUnit: The unit of the temperature to convert to ("C" for Celsius, "F" for Fahrenheit).

Inside the function, a loop iterates through each temperature value in the range.

Depending on the conversion direction (from Celsius to Fahrenheit or vice versa), the appropriate conversion formula is applied.

The converted temperature values are stored in an array (convertedTemps) for each input value.

The function returns the array of converted temperatures.

Step 4: Testing the Custom Formula

Enter temperature values in column A starting from row 2.

Use the custom formulas in cells B2 and C2 to convert the temperatures from Celsius to Fahrenheit and from Fahrenheit to Celsius, respectively.

For example, if you have entered the temperature values in column A, the corresponding converted temperatures in Celsius (in column C) and Fahrenheit (in column B) should be displayed.

Remember to enable the "Google Apps Script" extension and use the exact function name (CONVERT_TEMPERATURES) in your formulas.

GENERATE_PASSWORD Generate Secure Random Passwords with Google Apps Script

<https://youtu.be/qM3-KGtprbM>

In this example, we'll create a custom formula that generates a random password based on certain criteria.

Scenario: You want to create a custom formula that generates a random password with a specified length and a combination of uppercase letters, lowercase letters, numbers, and special characters.

	A	B
1	Password Length	Generated Password
2	12	2i%X0PmShN67
3	8	K^g7BBH3
4	15	dHNIqJ%nVNUAEg1

```
function GENERATE_PASSWORD(len) {  
  const charset =  
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()";
```

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

```

let password = '';
for(let i=0;i<len;i++){
const randomIndex =
Math.floor(Math.random()*charset.length);
password += charset.charAt(randomIndex);
}
return password;
}

```

The provided Google Apps Script function, named GENERATE_PASSWORD, is designed to generate a random password of a specified length. This function takes one parameter:

len: An integer representing the desired length of the generated password.

Here's a detailed description of how the function works:

```
const charset =
```

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#
```

```
$$%^&*()";
```

This line defines a character set that the password will be generated

from. The character set includes uppercase and lowercase letters, numbers (0-9),

and a selection of special characters such as !, @, #, \$, %, ^, & and *.

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

`let password = '';` This line initializes an empty string called `password`, which will be used to build the generated password.

The function uses a for loop to iterate `len` times. This loop is responsible for generating the password with the specified length:

```
for (let i = 0; i < len; i++) {  
  // ...  
}
```

Inside the loop, a random character is selected from the `charset` string. This is done by generating a random index between 0 and the length of the `charset` string:

```
const randomIndex = Math.floor(Math.random() * charset.length);
```

The `Math.random()` function generates a random decimal between 0 and 1, which is then multiplied by the length of the `charset`. `Math.floor()` is used to round this result down to an integer, ensuring it is a valid index.

The character at the random index in the `charset` string is added to the `password` string:

```
password += charset.charAt(randomIndex);
```

This step appends the randomly selected character to the `password` string.

After the loop has run `len` times and the `password` string has been constructed with random characters, the function returns the generated password.

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

To use this function, you can call it with the desired length and store the result in a variable:

```
const generatedPassword = GENERATE_PASSWORD(12);
```

The generatedPassword variable will contain a randomly generated password with a length of 12 characters, based on the characters in the charset. This function is useful for creating secure, randomized passwords for various applications.

Data Table:

A	B	C	D	E
Password Length	Generated Password			
12				
8				
15				
Total				

Step 1: Setting up the Spreadsheet

Create a new Google Sheets document.

Enter your password length values in column A starting from row 2.

Leave cells in column B empty for now.

In cell B1, enter "Generated Password" as a header.

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

Step 2: Writing the Google Apps Script Code

Click on "Extensions" in the top menu, then select "Apps Script".

Delete any default code and replace it with the following script:

```
// Custom formula to generate a random password
function GENERATE_PASSWORD(length) {
  var charset =
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01
  23456789!@#$%^&*(";
  var password = "";

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

  return password;
}
```

Step 3: Using the Custom Formula in Google Sheets

Go back to your Google Sheets document.

In cell B2, enter the following formula:

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

=GENERATE_PASSWORD(A2)

Explanation of the Code:

The function GENERATE_PASSWORD generates a random password with a specified length.

It takes one parameter: length - the length of the password to be generated.

Inside the function, a charset string is defined. This string contains all the possible characters that can be used in the password.

A loop iterates length times. In each iteration, a random index within the charset string is generated, and the character at that index is added to the password string.

The function returns the generated random password.

Step 4: Testing the Custom Formula

Enter password length values in column A starting from row 2.

Use the custom formula in cell B2 (or any other appropriate cell) to generate a random password based on the specified length.

For example, if you have entered the password length values as shown in the data table, the generated random passwords should appear in column B, each with the corresponding length specified in column A.


Remember to enable the "Google Apps Script" extension and use the exact function name (GENERATE_PASSWORD) in your formula.

Optimize Cost Calculations with Google Apps Script: Learn How!

https://youtu.be/YPP_hzF1Txw

In this example, we'll create a custom formula that calculates the total cost of items in a shopping cart, considering discounts based on a coupon code.

Scenario: You want to create a custom formula that calculates the total cost of items in a shopping cart, taking into account discounts based on a coupon code.

We're diving into the world of cost calculations, discounts, and coupon codes using Google Apps Script. 

Introducing the "COST_PER_ITEM" function, a handy tool that allows you to effortlessly determine the cost of an item based on its quantity, price, and even the potential discounts from coupon codes. Whether you're a business owner optimizing your e-commerce site or a developer looking to streamline your financial calculations, this function can be a game-changer.

Here's what you'll discover in this video:

How to use the "COST_PER_ITEM" function effectively.

Understanding the role of parameters like quantity, price, and coupon codes.

The importance of the "discountTable" and how it drives cost reductions.

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

A closer look at the "getDiscount" function to find applicable discounts. Plus, we'll walk you through practical examples and real-world applications to illustrate how this script can simplify cost calculations in various contexts.

Don't miss out on this opportunity to enhance your financial modeling and streamline cost management. Join us in the video to unlock the full potential of these functions.

#GoogleAppsScript #CodingTutorial #FinancialModeling #Discounts

#CouponCodes #CostCalculation #ProductivityTips #YouTubeTutorials

E2 | fx =COST_PER_ITEM(A2,B2,C2,D2,G2:H4)

	A	B	C	D	E	F	G	H
1	Item	Quantity	Price	Coupon Code	Cost		Coupon Code	Discount%
2	Item 1	2	\$12.00	SALE20	\$19.20		SALE20	0.2
3	Item 2	4	\$25.00	WELCOME	\$20.00		SUMMER	0.5
4	Item 3	1	\$20.00	WELCOME	\$4.00		WELCOME	0.8

```
=COST_PER_ITEM(A2,B2,C2,D2,G2:H4)
```

```
function
```

```
COST_PER_ITEM(item,quantity,price,couponCode,discountTable){  
  const discount = getDiscount(couponCode,discountTable);  
  let cost = quantity * price;  
  cost = cost - (cost * discount);  
  return cost;  
}
```

```
function getDiscount(couponCode,discountTable){
```

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


```
for(let i=0;i<discountTable.length;i++){  
  if(discountTable[i][0]===couponCode){  
    return discountTable[i][1];  
  }  
}  
return 0;  
}
```

The provided Google Apps Script code contains two functions: `COST_PER_ITEM` and `getDiscount`. These functions work together to calculate the cost of an item based on its quantity, price, and an optional coupon code that may offer a discount. Here's a detailed description of how these functions work:

1. `COST_PER_ITEM(item, quantity, price, couponCode, discountTable)` Function:

This function calculates the cost of an item, taking into account its quantity, price, and any applicable discounts provided through a coupon code. Here's a breakdown of its functionality:

- `item`: This parameter seems to be unused in the function, so it's not impacting the calculation of the cost.
- `quantity`: The quantity of the item you want to purchase.
- `price`: The unit price of the item.
- `couponCode`: An optional parameter that represents a coupon code. If a valid coupon code is provided, the function checks for discounts in the `discountTable`.

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

- `discountTable`: An array that stores coupon codes and their corresponding discounts. The array is a two-dimensional array, where each sub-array contains a coupon code in the first element and the associated discount percentage in the second element.

The function proceeds as follows:

- It calls the `getDiscount` function to check if a valid discount is available for the given `couponCode` in the `discountTable`.
- The `discount` variable is assigned the discount percentage obtained from the `getDiscount` function.
- It calculates the initial cost of the items ($\text{quantity} * \text{price}$).
- The cost is adjusted by subtracting the discount percentage from it ($\text{cost} - \text{cost} * \text{discount}$). If no valid discount is available, the discount percentage is 0, so the cost remains unchanged.
- Finally, the function returns the calculated cost after taking any discounts into account.

2. `getDiscount(couponCode, discountTable)` Function:

This function is responsible for looking up the discount associated with a given `couponCode` in the `discountTable`. Here's how it works:

- `couponCode`: The coupon code for which you want to find the discount.
- `discountTable`: The array that stores coupon codes and their corresponding discounts.

The function proceeds as follows:

- It iterates through the `discountTable` array using a for loop.

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

- For each entry in the discountTable, it checks if the coupon code provided as the argument matches the coupon code in the current entry (discountTable[i][0]).
- If a match is found, the function returns the discount percentage (discountTable[i][1]) associated with that coupon code.
- If no match is found after iterating through the entire discountTable, the function returns 0, indicating that there is no applicable discount for the provided coupon code.

These functions can be used to calculate the cost of items with discounts based on coupon codes, making them useful for various e-commerce and business scenarios where cost calculations need to consider discounts.

Title: "Unlocking Cost Optimization with Google Apps Script: A Comprehensive Guide"

In our latest video, we're delving deep into the world of cost calculations, discounts, and the magic of Google Apps Script. Whether you're a business owner striving to fine-tune your e-commerce platform or a developer seeking to streamline financial computations, this video is for you.

Join us to explore:

- ◆ The "COST_PER_ITEM" function: How to utilize it effectively.
- ◆ An in-depth look at key parameters like quantity, price, and coupon codes.
- ◆ The pivotal role of the "discountTable" in driving cost savings.

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

- ◆ A detailed understanding of the "getDiscount" function for identifying discounts.

We'll illustrate these concepts with real-world examples, making it easy to grasp the practical applications of this script in different scenarios.

Don't miss out on this opportunity to elevate your financial modeling and make cost management a breeze. Join us in the video to harness the full potential of these functions.

#GoogleAppsScript #FinancialModeling #Discounts #CostCalculation

#LinkedInLearning #BusinessOptimization #ProductivityTips #CodingTutorials

Item	Quantity	Price	Coupon Code	Cost		Coupon Code	Discount%
Item 1	2	\$12.00	SALE20	\$19.20		SALE20	0.2
Item 2	4	\$25.00	WELCOME	\$20.00		SUMMER	0.5
Item 3	1	\$20.00	WELCOME	\$4.00		WELCOME	0.8

Data Table:

A	B	C	D	E	F
---	---	---	---	---	---

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

Item	Quantity	Price	Coupon Code	Discount	Total Cost
Item 1	2	10	SALE20		
Item 2	3	15	SUMMER		
Item 3	1	20	WELCOME		
Total					

Coupon Codes and Discounts:

H	I
Coupon Code	Discount Percentage
SALE20	0.20
SUMMER	0.15
WELCOME	0.10

Step 1: Setting up the Spreadsheet

Create a new Google Sheets document.

Enter your item details, quantities, prices, and coupon codes in columns A to D starting from row 2.

Leave cells in columns E and F empty for now.

Enter the coupon codes and their corresponding discounts in columns H and I.

Step 2: Writing the Google Apps Script Code

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

Click on "Extensions" in the top menu, then select "Apps Script".

Delete any default code and replace it with the following script:

```
// Custom formula to calculate the total cost of items
with coupon discounts
function CALC_TOTAL_COST(items, quantities, prices,
couponCodes, discountTable) {
  var totalCost = 0;

  for (var i = 0; i < items.length; i++) {
    var quantity = quantities[i];
    var price = prices[i];
    var couponCode = couponCodes[i];
    var discount = getDiscount(couponCode,
discountTable);

    var cost = quantity * price;
    cost = cost - (cost * discount);

    totalCost += cost;
    items[i][5] = cost;
  }

  return totalCost;
}
```

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

```

}

// Helper function to get the discount from the
discount table
function getDiscount(couponCode, discountTable) {
  for (var i = 0; i < discountTable.length; i++) {
    if (discountTable[i] === couponCode) {
      return discountTable[i][1];
    }
  }
  return 0; // Default to no discount if coupon code
not found
}

```

Step 3: Using the Custom Formula in Google Sheets

Go back to your Google Sheets document.

In cell F2, enter the following formula:

=CALC_TOTAL_COST(A2:A, B2:B, C2:C, D2:D, H2:I)

Explanation of the Code:

The function CALC_TOTAL_COST calculates the total cost of items in a shopping cart, considering discounts based on a coupon code.

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

It takes five parameters: items, quantities, prices, couponCodes, and discountTable.

- items: The range of cells containing the item names.
- quantities: The range of cells containing the quantities for each item.
- prices: The range of cells containing the prices for each item.
- couponCodes: The range of cells containing the coupon codes for each item.
- discountTable: The range of cells containing the coupon codes and their corresponding discounts.

Inside the function, a loop iterates through each item in the shopping cart. It calculates the cost of each item and applies the discount based on the coupon code.

The getDiscount helper function is used to retrieve the discount from the discount table based on the coupon code.

The calculated cost of each item is stored in column F.

The function returns the total cost of all items in the shopping cart.

Step 4: Testing the Custom Formula

Enter item details, quantities, prices, and coupon codes in columns A to D starting from row 2.

Enter the coupon codes and their corresponding discounts in columns H and I.

Use the custom formula in cell F2 to calculate the total cost of items in the shopping cart with applicable discounts.

For example, if you have entered the item details and coupon codes as shown in the data table, the calculated total cost in cell F2 should reflect the cost of items with applicable discounts based on the coupon codes.

Remember to enable the "Google Apps Script" extension and use the exact function name (CALC_TOTAL_COST) in your formula.

Unlock the Power of Weighted Averages in Google Sheets with Google Apps Script

<https://youtu.be/uWTp7jX-3T8>

In this example, we'll create a custom formula that calculates the weighted average of a set of values based on their corresponding weights.

Scenario: You want to create a custom formula that calculates the weighted average of a set of values using their corresponding weights.

Unlock the Power of Weighted Averages in Google Sheets with Google Apps Script

Are you tired of manually calculating weighted averages in your Google Sheets documents? Say goodbye to tedious number crunching! In this video, we'll introduce you to a custom Google Apps Script that simplifies the process and automates the calculation of weighted averages for your data.

Here's what we'll cover in this video:

- The WEIGHTED_AVERAGE function: Learn how to set up and use this custom function to calculate weighted averages.

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

- Practical examples: We'll walk you through real-world scenarios, such as calculating the weighted grade for a student's scores or the weighted average of product sales data.
- Error handling: Discover how this script gracefully handles non-numeric values and zero weights, ensuring accurate calculations.

You don't need to be a coding expert to benefit from this script; we'll guide you through the setup process step by step. Get ready to save time and streamline your data analysis in Google Sheets.

C2 fx =WEIGHTED_AVERAGE(A2:A4,B2:B4)

	A	B	C
1	Value	Weight	Total
2	99.00	0.50	49.50
3	70.00	0.30	21.00
4	95.00	0.30	28.50
5	264.00		99.00

```
function WEIGHTED_AVERAGE(values1,weights1){
const weightedAverage = [];
for(let i=0;i<values1.length;i++){
const val = calWeightAver(values1[i],weights1[i]);
weightedAverage.push([val]);
}
```

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

```

}
return weightedAverage;
}

function calculateWeightedAverage(value,weight){
if(!isNaN(value) && !isNaN(weight) && weight !== 0 ){
return value * weight;
}else{
return 0;
}
}
}

```

=WEIGHTED_AVERAGE(A2:A4,B2:B4)

The provided Google Apps Script code defines two functions,

WEIGHTED_AVERAGE and calculateWeightedAverage, which work together to calculate the weighted average for a set of values and their corresponding weights. Below is a detailed explanation of how these functions work:

1. calculateWeightedAverage(value, weight) Function:

This is a utility function that calculates the weighted value of a single pair of value and weight. It ensures that both the value and weight are numeric (not NaN), and that the weight is not zero to avoid division by zero errors. The function can be used for individual value-weight pairs. If the provided values or weights don't

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

meet these criteria, it returns a default value of 0 (or any other specified value) to handle the error.

- value: A numeric value.
- weight: A numeric weight associated with the value.

2. WEIGHTED_AVERAGE(values1, weights1) Function:

This function calculates the weighted average for a set of values and their corresponding weights. It accepts two arrays as parameters:

- values1: An array containing values for which you want to calculate the weighted average.
- weights1: An array containing weights corresponding to the values in values1.

Here's how the WEIGHTED_AVERAGE function works:

- It initializes an empty array called weightedAverage1 to store the calculated weighted values for each pair of value and weight.
- It uses a for loop to iterate through the values1 array. For each element at index i, it calls the calculateWeightedAverage function, passing the value at the current index and the corresponding weight. The calculated weighted value is then stored in the weightedAverage1 array as an element within its own array.
- Finally, it returns the weightedAverage1 array, which contains the calculated weighted values for the set of values in values1 and their corresponding weights in weights1.

In summary, these functions are designed to calculate the weighted average for a set of values and their corresponding weights while ensuring that non-numeric or zero weights do not disrupt the calculation. The calculateWeightedAverage

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

function provides a basic building block for these calculations, and the `WEIGHTED_AVERAGE` function applies it to an entire array of value-weight pairs.

Value	Weight	Total
99.00	0.50	49.50
70.00	0.30	21.00
95.00	0.30	28.50
264.00		99.00

Step 1: Access Google Sheets

Open a web browser and navigate to Google Sheets at sheets.google.com.

Log in to your Google account or create one if you don't have an account.

Step 2: Create or Open a Google Sheets Document

Create a new Google Sheets document by clicking the "+ Blank" option or open an existing document where you want to calculate the weighted averages.

Step 3: Open the Script Editor

In the Google Sheets document, click on "Extensions" in the top menu.

Select "Apps Script" from the dropdown menu.

Step 4: Copy and Paste the Script

In the Apps Script editor, you will see a default `myFunction()` function. You can remove this function if you don't need it.

Copy the provided script that contains the `WEIGHTED_AVERAGE` and `calWeightAver` functions.

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

Paste the script into the script editor, replacing any existing content.

Step 5: Save the Project

Click the floppy disk icon (or press Ctrl + S or Cmd + S) to save the project.

Provide a name for your project, if prompted, and click "OK."

Step 6: Close the Script Editor

Close the Apps Script editor tab.

Step 7: Use the WEIGHTED_AVERAGE Function in Google Sheets

In your Google Sheets document, you can use the WEIGHTED_AVERAGE function in a cell to calculate the weighted average for a set of values and their corresponding weights.

For example, if you have values in cells A1:A5 and weights in cells B1:B5, you can use the formula =WEIGHTED_AVERAGE(A1:A5, B1:B5) in another cell to get the weighted average.

Step 8: Test the Function

Enter values and weights in the specified cells.

Place the WEIGHTED_AVERAGE formula in a cell.

The cell with the formula will display the calculated weighted average.

You've now successfully set up and used the provided script in Google Sheets to calculate weighted averages. This script will calculate the weighted average for a set of values and their corresponding weights, while handling cases with non-numeric or zero weights gracefully.

CONVERT_TO_LETTER_GRADE

<https://youtu.be/xPn4ToAQAlw>

In this example, we'll create a custom formula that converts a numeric grade into a letter grade using a custom grading scale.

Scenario: You want to create a custom formula that converts a numeric grade into a letter grade using a custom grading scale.

Unlock the Magic of Automated Grade Conversion in Google Sheets! 🧙‍♂️ ✨

Tired of manually converting numeric grades to letter grades in your Google

Sheets? We've got the ultimate solution for you! In our latest video, we'll

introduce you to a custom Google Apps Script that takes the tedium out of grading and automates the process for you.



Here's what you'll discover in this video:



✓ The CONVERT_TO_LETTER_GRADE function: Learn how to use this powerful function to convert numeric grades to letter grades for an entire list of students or assignments. ✓ Real-world applications: We'll demonstrate how this script can be applied in various scenarios, such as grading students' scores or converting test results. ✓ Customizable grade scales: You can set up your own grade scale to suit your specific grading system or academic institution.

You don't need to be a coding expert to implement this script; we'll guide you step by step through the setup process. It's a game-changer for educators, students, and anyone managing grades in Google Sheets.

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

Don't miss this opportunity to simplify your grading and save time! Join us for this essential tutorial.

#GoogleAppsScript #GoogleSheets #GradeConversion #Education #Automation
#CodingTutorial #YouTubeTutorials #EdTech #ProductivityTips #Academics

	A	B	C	D	E	F
1	Numeric Grade	Letter Grade			Lower Limit	Letter Grade
2	95	A			85	A
3	80	B			70	B
4	63	C			60	C
5					50	D
6					0	F

```
function CONVERT_TO_LETTER_GRADE(numGrades,gradeScale){
  const letterGrades = [];
  for(let i=0;i<numGrades.length;i++){
    const numGrade = numGrades[i];
    const letterGrade = getLetterGrade(numGrade,gradeScale);
    letterGrades.push([letterGrade]);
  }
  return letterGrades;
}
```

```
function getLetterGrade(numGrade,gradeScale){
  for(let i=0;i<gradeScale.length;i++){
```

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


```
if(numGrade>=gradeScale[i][0]){  
  return gradeScale[i][1];  
}  
}  
return "";  
}
```

The provided Google Apps Script code defines two functions, `CONVERT_TO_LETTER_GRADE` and `getLetterGrade`, which work together to convert numeric grades into letter grades based on a specified grade scale. Below is a detailed explanation of how these functions work:

1. `getLetterGrade(numGrade, gradeScale)` Function:

This function is responsible for converting a numeric grade (`numGrade`) into its corresponding letter grade based on the provided grade scale (`gradeScale`). The function works as follows:

- It starts by initializing an empty string to store the letter grade.
- It uses a for loop to iterate through the `gradeScale` array. The `gradeScale` array is expected to contain pairs of numeric grade thresholds and their corresponding letter grades.
- For each iteration, it checks if `numGrade` is greater than or equal to the numeric grade threshold in the `gradeScale`. If the condition is met, it returns the corresponding letter grade.

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

- If no match is found in the gradeScale after looping through all the thresholds, the function returns an empty string, indicating that no letter grade was found for the given numeric grade.

2. CONVERT_TO_LETTER_GRADE(numGrades, gradeScale) Function:

This function is designed to convert an array of numeric grades (numGrades) into an array of their corresponding letter grades using the getLetterGrade function.

The function works as follows:

- It initializes an empty array called letterGrades to store the resulting letter grades.
- It uses a for loop to iterate through the numGrades array. For each numeric grade in the array, it calls the getLetterGrade function to obtain the corresponding letter grade based on the provided gradeScale. The letter grade is then stored in the letterGrades array as an element within its own array.
- Finally, it returns the letterGrades array, which contains the converted letter grades corresponding to the input numeric grades.

In summary, these functions are used to convert numeric grades to letter grades based on a given grade scale. The getLetterGrade function is responsible for finding the appropriate letter grade based on the numeric grade and the provided scale, while the CONVERT_TO_LETTER_GRADE function facilitates the conversion for an array of numeric grades.

Data Table:

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

Numeric Grade	Letter Grade
95	A
80	B
63	C

Custom Grading Scale:

E	F
Lower Limit	Letter Grade
90	A
80	B
70	C
60	D
0	F

Step 1: Setting up the Spreadsheet

Create a new Google Sheets document.

Enter the numeric grades in column A starting from row 2.

Leave cells in column B empty for now.

Enter the lower limits and corresponding letter grades in columns E and F.

Step 2: Writing the Google Apps Script Code

Click on "Extensions" in the top menu, then select "Apps Script".

Delete any default code and replace it with the following script:

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

```
// Custom formula to convert numeric grade to letter  
grade
```

```
function CONVERT_TO_LETTER_GRADE(numericGrades,  
gradeScale) {  
  var letterGrades = [];  
  
  for (var i = 0; i < numericGrades.length; i++) {  
    var numericGrade = numericGrades[i];  
    var letterGrade = getLetterGrade(numericGrade,  
gradeScale);  
    letterGrades.push([letterGrade]);  
  }  
  
  return letterGrades;  
}
```

```
// Helper function to get the letter grade based on the  
grade scale
```

```
function getLetterGrade(numericGrade, gradeScale) {  
  for (var i = 0; i < gradeScale.length; i++) {  
    if (numericGrade >= gradeScale[i]) {  
      return gradeScale[i][1];  
    }  
  }  
}
```

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

```
}  
  return ""; // Default to empty if grade is not found  
in scale  
}
```

Step 3: Using the Custom Formula in Google Sheets

Go back to your Google Sheets document.

In cell B2, enter the following formula:

=CONVERT_TO_LETTER_GRADE(A2:A, E2:F)

Explanation of the Code:

The function `CONVERT_TO_LETTER_GRADE` converts numeric grades to letter grades using a custom grading scale.

It takes two parameters: `numericGrades` and `gradeScale`.

- `numericGrades`: The range of cells containing the numeric grades to be converted.
- `gradeScale`: The range of cells containing the lower limits and corresponding letter grades for the grading scale.

Inside the function, a loop iterates through each numeric grade.

The `getLetterGrade` helper function is used to determine the letter grade based on the numeric grade and the grading scale.

The calculated letter grades are stored in the `letterGrades` array.

The function returns the array of converted letter grades.

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

Step 4: Testing the Custom Formula

Enter numeric grades in column A starting from row 2.

Enter the lower limits and corresponding letter grades in columns E and F.

Use the custom formula in cell B2 to convert the numeric grades into letter grades based on the custom grading scale.

For example, if you have entered the numeric grades and grading scale as shown in the data table, the corresponding letter grades should appear in column B.

Remember to enable the "Google Apps Script" extension and use the exact function name (CONVERT_TO_LETTER_GRADE) in your formula.

Custom formula to calculate average rating

<https://youtu.be/a86BXNi1zmw>

In this example, we'll create a custom formula that helps you analyze survey data by calculating the average rating for a set of responses.

Scenario: You want to create a custom formula that calculates the average rating from a set of survey responses, where each response is rated on a scale from 1 to 5.

E2 | fx =CALC_AVERAGE_RATING(A2:A4, B2:D4)

	A	B	C	D	E
1	Question	Response 1	Response 2	Response 3	Average Rating
2	Question 1	4	5	3	4
3	Question 2	2	4	5	3.666666667
4	Question 3	3	3	4	3.333333333

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

🌟 Simplify Your Data Analysis with Custom Google Apps Script for Average Ratings! 📊

Are you overwhelmed by the task of calculating average ratings from a multitude of responses to various questions? We've got you covered! In our latest video, we're introducing a powerful Google Apps Script that automates the process, making it a breeze to determine average ratings for each question.

📋 In this video, we'll cover:

✅ The `CALC_AVERAGE_RATING` function: Learn how to harness this custom script to effortlessly compute average ratings for multiple questions and responses. ✅

Real-life use cases: We'll walk you through practical examples, such as survey analysis, customer feedback assessments, and more. ✅

Streamlined insights: Discover how this script transforms raw data into meaningful statistics.

You don't need to be a coding expert to get started. We'll guide you step by step through the setup process, ensuring you can easily incorporate this automation into your data analysis workflow.

Say goodbye to manual number crunching and hello to efficient data analysis! Join us for this essential tutorial.

#GoogleAppsScript #DataAnalysis #Automation #AverageRatings

#YouTubeTutorials #DataInsights #SurveyAnalysis #CustomerFeedback

#Productivity #CodingTutorial

```
function CALC_AVERAGE_RATING(questions, responses) {  
  const averageRatings = [];
```

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

```

for(let i=0;i<questions.length;i++){
let totalRating = 0;
for(let j=0;j<responses[i].length;j++){
totalRating += responses[i][j];
}
const averageRating = totalRating / responses[i].length;
averageRatings.push([averageRating]);
}
return averageRatings;
}

```

The provided code defines a Google Apps Script function named `CALC_AVERAGE_RATING`. This function is designed to calculate the average ratings for a list of questions based on corresponding responses. Here's a detailed explanation of how the code works:

Function Name: `CALC_AVERAGE_RATING(questions, responses)`

This function takes two parameters as input:

- `questions`: An array that contains the questions for which you want to calculate average ratings.
- `responses`: A 2D array where each sub-array represents the responses to a specific question. The outer array contains all the questions, and each sub-array contains the responses for a particular question.

Initialization:

Inside the function, an empty array named `averageRatings` is initialized. This array will store the calculated average ratings for each question.

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

Loop Through Questions:

The function uses a for loop to iterate through each question in the questions array. The loop index *i* represents the current question being processed.

Calculate Total Rating:

Within the outer loop, there is an inner loop that iterates through the responses for the current question (retrieved using `responses[i]`). The loop index *j* represents the current response being considered. The `totalRating` variable is used to accumulate the sum of all the responses for the current question.

Calculate Average Rating:

After iterating through all the responses for the current question, the code calculates the average rating for that question by dividing the `totalRating` by the number of responses (which is the length of the `responses[i]` array). The average rating is then stored in an array within the `averageRatings` array.

Return the Results:

After processing all questions and their responses, the function returns the `averageRatings` array, which contains the average rating for each question.

In summary, this script is helpful for conducting surveys or assessments where multiple respondents provide ratings for various questions. It calculates the average rating for each question, allowing you to analyze the overall response for each item on your list of questions. The function provides a simple way to obtain insights into how questions are rated on average based on collected responses.

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

Data Table:

A	B	C	D	E
Question	Response 1	Response 2	Response 3	Average Rating
Question 1	4	5	3	
Question 2	2	4	5	
Question 3	3	3	4	
Total				

Step 1: Setting up the Spreadsheet

Create a new Google Sheets document.

Enter the survey questions in column A starting from row 2.

Enter the responses for each question in columns B to D.

Leave cells in column E empty for now.

Step 2: Writing the Google Apps Script Code

Click on "Extensions" in the top menu, then select "Apps Script".

Delete any default code and replace it with the following script:

```
// Custom formula to calculate average rating
function CALC_AVERAGE_RATING(questions, responses) {
  var averageRatings = [];
```

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

```

for (var i = 0; i < questions.length; i++) {
  var totalRating = 0;

  for (var j = 0; j < responses[i].length; j++) {
    totalRating += responses[i][j];
  }

  var averageRating = totalRating /
responses[i].length;
  averageRatings.push([averageRating]);
}

return averageRatings;
}

```

Step 3: Using the Custom Formula in Google Sheets

Go back to your Google Sheets document.

In cell E2, enter the following formula:

=CALC_AVERAGE_RATING(A2:A, B2:D)

Explanation of the Code:

The function CALC_AVERAGE_RATING calculates the average rating for a set of survey responses.

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

It takes two parameters: questions and responses.

- questions: The range of cells containing the survey questions.
- responses: The range of cells containing the responses for each question.

Inside the function, a loop iterates through each question.

Within the question loop, another loop iterates through each response for the current question.

The total rating for each question is calculated by summing up all the responses.

The average rating for each question is calculated by dividing the total rating by the number of responses.

The calculated average ratings are stored in the averageRatings array.

The function returns the array of calculated average ratings.

Step 4: Testing the Custom Formula

Enter survey questions in column A starting from row 2.

Enter responses for each question in columns B to D.

Use the custom formula in cell E2 to calculate the average rating for each question.

For example, if you have entered the survey responses as shown in the data table, the calculated average ratings for each question should appear in column E.

Remember to enable the "Google Apps Script" extension and use the exact function name (CALC_AVERAGE_RATING) in your formula.