Back in your event listener, you need to update the text of the result element. You can use a ternary operator to achieve this task.

Here is an example of assigning the result of a ternary operator to an element's text content:

Example Code

el.textContent = condition ? "Use this text if the condition is true" : "Use this text if the condition is false";

After the if statement, use a ternary operator to check the truthiness of calling isSpam() with messageInput.value as the argument. If true, set the textContent property on the result element to "Oh no! This looks like a spam message.". Otherwise, set it to "This message does not seem to contain any spam."

Then set the messageInput element's value property to an empty string.

Regular expressions can take flags to modify their behavior. For instance, the i flag can be used to make the expression ignore case, causing it to match hello, HELLO, and Hello for the expression /hello/.

Flags are added after the trailing slash. Add the i flag to your helpRegex.

Strings have a .match() method, which accepts a regular expression as an argument and determines if the string matches that expression.

Update your isSpam() function to implicitly return the result of calling the .match() method on msg, passing helpRegex as the argument.

Then, try entering some messages on your page and see the result.

**Step 12**

Arrays have a .some() method. Like the .filter() method, .some() accepts a callback function which should take an element of the array as the argument. The .some() method will return true if the callback function returns true for at least one element in the array.

Here is an example of a .some() method call to check if any element in the array is an uppercase letter.

Example Code

const arr = ["A", "b", "C"];

arr.some(letter => letter === letter.toUpperCase());

Use the .some() method to check if testing your msg on any of your denyList regular expressions returns true.

Use regex as the parameter for the callback function, for clarity.

You need to match a number before the text dollars. While you could write out 0|1|2 and so on, regular expressions have a feature that makes this easier.

A character class is defined by square brackets, and matches any character within the brackets. For example, [aeiou] matches any character in the list aeiou. You can also define a range of characters to match using a hyphen. For example, [a-z] matches any character from a to z.

Add a character class to match the digits 0 through 9 to your dollarRegex expression - remember the digit must come before the word dollars, and there should be a space between the digit and the word.

A capture group is a way to define a part of the expression that should be captured and saved for later reference. You can define a capture group by wrapping a part of your expression in parentheses. For example, /h(i|ey) camper/ would match either hi camper or hey camper, and would capture i or ey in a group.

Turn your place values into a capture group.

Now that you have your capture group, you can mark the entire pattern as an optional match. The ? quantifier matches zero or one occurrence of the preceding character or group. For example, the regular expression /colou?r/ matches both color and colour, because the u is optional.

Mark your capture group as optional.

One last thing with this expression. You don't actually need the match value from your capture group, so you can turn it into a non-capturing group. This will allow you to group the characters together without preserving the result.

To create a non-capturing group in a regular expression, you can add ?: after the opening parenthesis of a group. For instance, (?:a|b) will match either a or b, but it will not capture the result.

Update your regular expression to use a non-capturing group.

# Step 25

Your regex should match whole words, not partial words. That is, you do not want to match hands-free money management.

To do this, start by checking for spaces before and after your pattern. You can do this by using the meta character \s, which will match spaces, tabs, and line breaks.

You still aren't matching free money yet, because you need to match the end of the string as well.

Like the ^ anchor, you can use the $ anchor to match the end of the string.

Update your regular expression to match either the end of the string or a space, like you did for the beginning of the string.

If you try entering the message free money, you'll notice it doesn't match your expression! This is because \s doesn't match the beginning or end of the text.

To match the beginning of the text, you can use the ^ anchor. This asserts that your pattern match starts at the beginning of the full string.

Replace your first \s character with a non-capturing group that matches \s or ^.