Module 10 Assignment

Create a Word Cloud

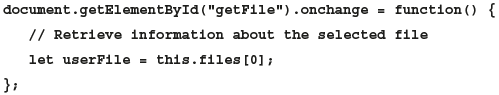
**Directions**

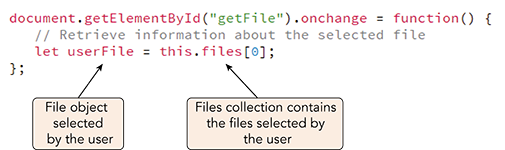
You will complete this assignment in a series of steps. Read each step carefully.

**Part 1. Create an onchange event handler:**

Add an onchange event handler to the getFile input box to retrieve information about the first (and only) file selected by the user. Use the this keyword to reference the input box that initiated the change event and store information about the file in the userFile variable.

1. Download the m10 folder from https://github.com/Western-Wyoming-Computer-Science/COSC1350 and save it to your COSC1350 folder.
2. Open the COSC1350 > m10 folder in Visual Studio Code, then open the js07\_txt.html and js07\_txt.js files. Enter your name and the date in the comment section of each file and then save them as js07.html and js07.js, respectively.
3. Return to the js07.html file in your code editor. Within the head section, add a script element to run the js07.js script, deferring the loading the script file until after the entire page has loaded.
4. Directly below the initial comment section, insert the following code to run an anonymous function in response to the change event (see Figure 1):





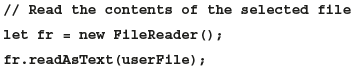
**Figure 1. Retrieving a selected file**

Having stored information about the selected file, you will next read its contents.

Next you will create a file reader for the word cloud script to load and read the contents of the userFile document. Once the load event for the document has occurred (indicating a successful reading), create an onload event handler to write the contents of the userFile document to the web page.

**Part 2. Load a document using the file reader:**

1. Within the anonymous function add the following commands to create a FileReader object and to load the contents of the userFile object as a text file.



1. Next, apply the onload event handler to the file reader so when the document is complete and successfully loaded, its contents will be written to the web page.

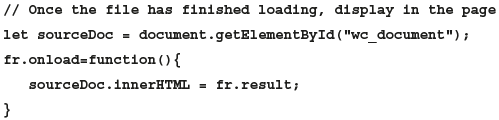
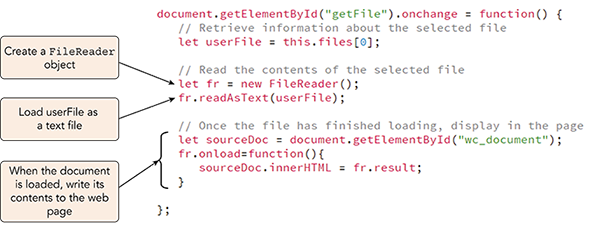


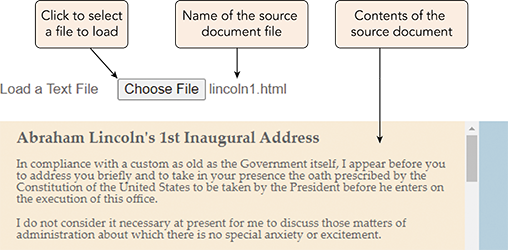
Figure 2 describes the newly added code to the anonymous function.



**Figure 2. Reading and loading the contents of a text file**

1. Save your changes to the file and then load **js07.html** in your browser.
2. Click the **Choose File** button to open the File Open dialog box.
3. Locate and open the **lincoln1.html** file from the m10 folder.

As shown in Figure 3, the contents of Lincoln’s first inaugural address should appear within the web page.



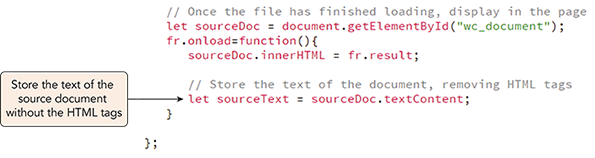
**Figure 3. External text loaded into the Word Cloud app**

**Part 3. Create the sourceText variable:**

In this assignment you will use the lincoln1.html file to develop and test the Word Cloud app. The content of that file includes HTML tags marking the speech’s main heading and paragraphs. HTML tags are not included in a word cloud, so you will strip them out, storing only the text content of the speech in a variable named sourceText.

1. Return to the **js07.js** file in your code editor.
2. Within the anonymous function for the onload event handler, add the following code to store the text of the source document (see Figure 4).

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Store the text of the document, comma, removing HTML tags. Line 2. let source Text, equals, source Doc, dot, text Content, semi-colon.



**Figure 4. Extracting text content from the source document**

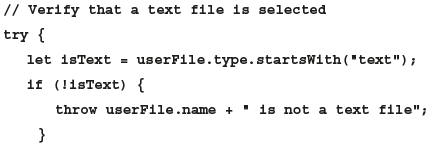
1. Save your changes to the file, load **js07.html** in your browser and then load the **lincoln1.html** file into the web page.
2. Open your browser’s debugger console and confirm that no errors are reported by the debugger.

**Part 4. Test that the user has selected a text file:**

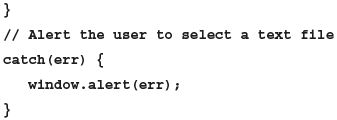
The Word Cloud app needs to confirm that the user has selected a text file for processing and not a non-text file like an image or video. The type property of the file object indicates the content of the file by returning file’s MIME type. Text files have MIME types starting with the substring “text”, such as “text/plain”, “text/html”, or “text/javascript”. Thus, you can confirm that the user selected a text file by checking whether the file’s MIME type starts with the “text” substring.

Add a try catch statement to the Word Cloud app that uses the startsWith() method to verify that a text file has been selected by the user, throwing an error if it has not.

1. Return to the **js07.js** file in your code editor.
2. Directly below the statement declaring the userFile variable, add the following initial code for a try catch statement:



1. Scroll down and directly after the onload anonymous function, insert the following code closing the try statement and adding a catch statement for catching the thrown error.



1. Indent the content of the try statement to make your code easier to read. Figure 5 describes the revised code in the file.

A code block with code that verifies that the user selects a text file. Program code. In the code, the words in the variable names are merged, and the code contains the following keywords: let, this, if, new. Line 1: document, dot, get Element By I d, left parenthesis, left double quotation mark, get File, right double quotation mark, right parenthesis, dot, on change, equals, function, left parenthesis, right parenthesis, left brace. Line 2, indented once: Forward slash, forward slash, Retrieve information about the selected file. Line 3, indented once: let, user File, equals, this, dot, files, left bracket, 0, right bracket, semicolon Line 4: Blank. Line 5, Indented once: Forward slash, forward slash, Verify that a text file is selected. Line 6, indented once: try, left brace. Line 7, indented twice: let, is Text, equals, user File, dot, type, dot, starts With, left parenthesis, left double quotation mark, text, right double quotation mark, right parenthesis, semicolon. Line 8, indented twice: if, left parenthesis, exclamation mark, is Text, right parenthesis, left brace. Line 9, indented 3 times: throw user File, dot, name, plus, left double quotation mark, character space, is not a text file, right double quotation mark, semicolon. Line 10, indented twice: Right brace. Line 11: Blank. Line 12, indented twice: Forward slash, forward slash, Read the contents of the selected file. Line 13, indented twice: let, f r, equals new, File Reader, left parenthesis, right parenthesis, semicolon. Line 14, indented twice: f r, dot, read As Text, left parenthesis, user File, right parenthesis, semicolon. Line 15: Blank. Line 16, indented twice: Forward slash, forward slash, Once the file has finished loading, display in the page. Line 17, indented twice: let, source Doc, equals, document, dot, get Element By I d, left parenthesis, left double quotation mark, w c underscore document, right double quotation mark, right parenthesis, semicolon. Line 18, indented twice: f r, dot, on load, equals, function, left parenthesis, right parenthesis, left brace. Line 19, indented 3 times: source Doc, dot, inner H T M L, equals, f r, dot result, semicolon. Line 20: Blank. Line 21, indented 3 times: Forward slash, forward slash, Store the text of the document, removing H T M L tags. Line 22, indented 3 times: let, source Text, equals, source Doc, dot, text Content, semicolon. Line 23, indented twice: Right brace. Line 24, indented once: Right brace. Line 25: Blank. Line 26, indented once: Forward slash, forward slash, Alert the user to select a text file. Line 27, indented once: catch, left parenthesis, e r r, right parenthesis, left brace. Line 28, indented twice: window, dot, alert, left parenthesis, e r r, right parenthesis, semicolon. Line 29, indented once: Right brace. Line 30: Blank. Line 31: Right brace, semicolon. Line 7 of the above code tests whether the source file's MIME type starts with text. In lines 8 to 10, if the MIME type does not start with text, an error is thrown. In lines 12 to 23, commands run if no error is thrown. In lines 26 to 28, commands run if an error is caught.

**Figure 5.** **Verifying that the user selects a text file**

1. Save your changes to the file and then load **js07.html** in your browser.
2. Click **Choose File** and select the **wordcloud.png** file. Verify that the page displays an alert box indicating that wordcloud.png is not a text file.
3. Click **Choose File** again and select **lincoln1.html**, verifying that the contents of that file load without error.

If your program does not work correctly, check your code against that shown in Figure 5. A common mistake is omitting an opening or closing curly brace within the try catch statement. Having confirmed that the user has selected a text file, you can begin building the commands that generate a word cloud for the document. You will start by modifying the text within the file.

**Part 5. Modify the source text:**

You will make the following changes to the text of the source document:

* Convert all characters to lowercase letters to remove the distinction between words like “Nation” and “nation”.
* Strip out any leading or trailing whitespace characters from the text so that the text begins and ends with a printable character.

To create a new string containing only lowercase characters apply the following toLowerCase() method to the sourceText variable:

Program code. In the code, the words in the variable names are merged. Line 1. source Text, equals, source Text, dot, to Lower Case, left parenthesis, right parenthesis, semi-colon.

To strip out leading and trailing whitespace characters apply the trim() function:

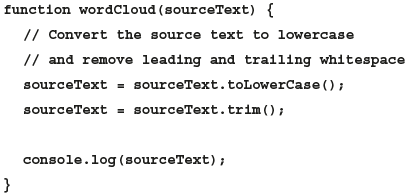
Program code. In the code, the words in the variable names are merged. Line 1. source Text, equals, source Text, dot, trim, left parenthesis, right parenthesis, semi-colon.

Add both of these commands to the wordCloud() function running them after the file is successfully read by the browser.

1. Return to the **js07.js** file in your code editor.
2. Directly after the statement declaring the sourceText variable, add the following command to call the wordCloud() function:

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Generate the word cloud. Line 2. word Cloud, left parenthesis, source Text, right parenthesis, semi-colon.

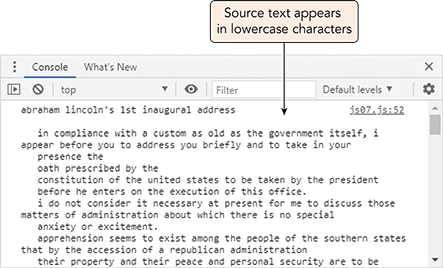
1. After the closing brace of the catch statement, add the following initial code for the wordCloud() function:



The app writes the content of the sourceText to the debugger console so you can view the changing value of that variable as you develop the Word Cloud app (see Figure 6).

  
  
  
**Figure 6. Modifying the source text**

1. Save your changes to the file and then reload **js07.html** in your browser and load **lincoln1.html** in the web page.
2. View the console log in the browser debugger and verify that content of the sourceText variable is displayed in lowercase characters (see Figure 7).



**Figure 7. Source text in the debugger console**

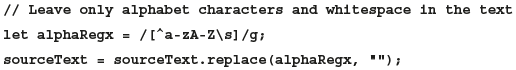
**Part 6. Remove non-alphabetic characters from the source text:**

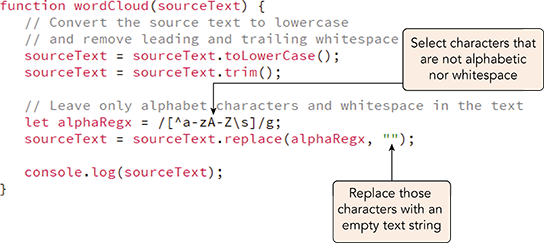
Use the replace() method to move all punctuation marks and digits from the Word Cloud app’s source text, replacing them with empty text strings. The regular expression to match all characters that are not alphabetic and not whitespace is:

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, left bracket, caret, a, hyphen, z A, hyphen, Z, backward slash, s, right bracket, forward slash, g.

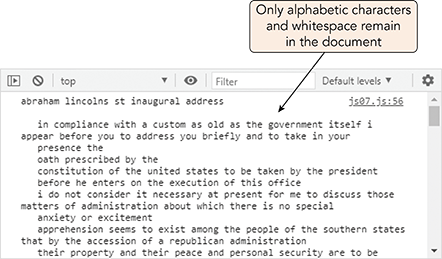
Add your code to the wordCloud() function now.

1. Return to the **js07.js** file in your code editor.
2. Go to the wordCloud() function and directly before the statement writing sourceText to the debugger console, add the following code as described in Figure 8:



  
**Figure 8. Removing non-alphabetic characters**

1. Save your changes to the file and then reload **js07.html** in your browser and load **lincoln1.html** in the app.
2. View the console log to confirm that the source text no longer contains punctuation marks or digits, but only alphabetic characters and blank spaces (see Figure 9).

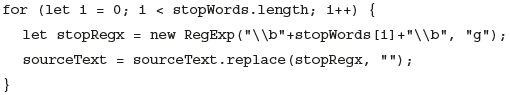
  
  
**Figure 9. Source text with only alphabetic characters**

**Part 7. Remove stop words from the source text:**

The Word Cloud app also needs to remove stop words from the source text. A stop word is a word not normally included within a word cloud because it provides no meaning, including articles such as “and”, “if”, “is”, and “the”. An array of stop words has been created for you and stored at the bottom of the js07.js file. Figure 10 shows a portion of the stopWords array.

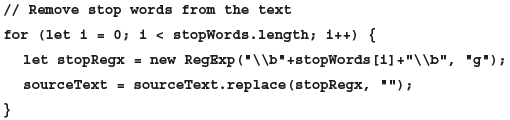
A code block with code that contains an array that stores a set of stop words that need to be removed from the source text for the word cloud app. Program code. In the code, the words in the variable names are merged, and the code contains the following keywords: let. Line 1: Forward slash, asterisk, a series of hyphens, asterisk, forward slash. Line 2: Forward slash, asterisk, Array of words to NOT include in the word cloud, asterisk, forward slash. Line 3: Forward slash, asterisk, a series of hyphens, asterisk, forward slash. Line 4: Blank. Line 5: let, stop Words, equals, left bracket, left double quotation mark, a, right double quotation mark, comma, left double quotation mark, about, right double quotation mark, comma, left double quotation mark, above, right double quotation mark, comma, left double quotation mark, across, right double quotation mark, comma, left double quotation mark, after, right double quotation mark, comma, left double quotation mark, afterwards, right double quotation mark, comma, left double quotation mark, again, right double quotation mark, comma, left double quotation mark, against, right double quotation mark, comma. Line 6, indented a few times: left double quotation mark, ago, right double quotation mark, comma, left double quotation mark, all, right double quotation mark, comma, left double quotation mark, almost, right double quotation mark, comma, left double quotation mark, alone, right double quotation mark, comma, left double quotation mark, along, right double quotation mark, comma, left double quotation mark, already, right double quotation mark, comma, left double quotation mark, also, right double quotation mark, comma, left double quotation mark, although, right double quotation mark, comma. Line 7: left double quotation mark, always, right double quotation mark, comma, left double quotation mark, am, right double quotation mark, comma, left double quotation mark, among, right double quotation mark, comma, left double quotation mark, amongst, right double quotation mark, comma, left double quotation mark, amongst, right double quotation mark, comma, left double quotation mark, amount, right double quotation mark, comma, left double quotation mark, an, right double quotation mark, comma, left double quotation mark, and, right double quotation mark, comma. Line 8: left double quotation mark, another, right double quotation mark, comma, left double quotation mark, any, right double quotation mark, comma, left double quotation mark, anyhow, right double quotation mark, comma, left double quotation mark, anyone, right double quotation mark, comma, left double quotation mark, anything, right double quotation mark, comma, left double quotation mark, anyway, right double quotation mark, comma, left double quotation mark, anywhere, right double quotation mark, comma. Line 9: left double quotation mark, are, right double quotation mark, comma, left double quotation mark, around, right double quotation mark, comma, left double quotation mark, as, right double quotation mark, comma, left double quotation mark, at, right double quotation mark, comma, left double quotation mark, back, right double quotation mark, comma, left double quotation mark, be, right double quotation mark, comma, left double quotation mark, became, right double quotation mark, comma, left double quotation mark, because, right double quotation mark, comma, left double quotation mark, become, right double quotation mark, comma. Line 10: left double quotation mark, becomes, right double quotation mark, comma, left double quotation mark, becoming, right double quotation mark, comma, left double quotation mark, been, right double quotation mark, comma, left double quotation mark, before, right double quotation mark, comma, left double quotation mark, beforehand, right double quotation mark, comma, left double quotation mark, behind, right double quotation mark, comma, left double quotation mark, being, right double quotation mark, comma.

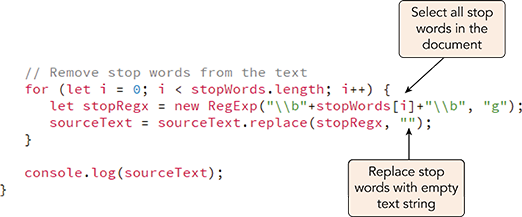
To remove the stop words from the source text, the Word Cloud app will examine every entry in the stopWords array and apply the replace() method to replace the stop word in the source text with an empty text string. The for loop is:



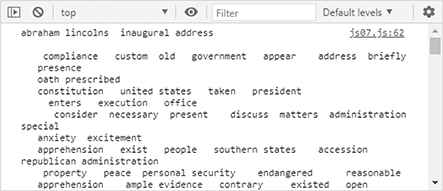
Note that the regular expression using \b to mark the word boundaries around each stop word and the global flag, g, to do the replacement throughout the source text. Add this for loop to the wordCloud() function now.

1. Return to the **js07.js** file in your code editor.
2. Directly before the statement writing sourceText to the debugger console, add the following for loop as described in Figure 11.



  
**Figure 11. Removing stop words from the source text**

1. Save your changes to the file and then reload **js07.html** in your browser and load **lincoln1.html** in the app.
2. View the console log to confirm that stop words have been removed from the source text (see Figure 12).

  
  
**Figure 12. Source text without stop words**

Having removed the stop words from the source text, you will next place the remaining words into an array.

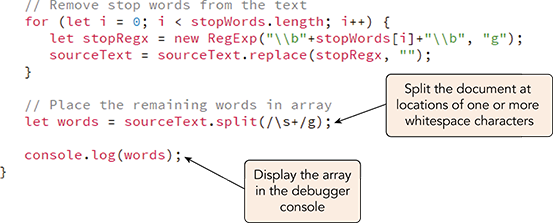
**Part 8. Place the source text words into an array:**

Use the split() method now to split the source text as every occurrence of one or more whitespace characters, creating an array of words from the source text.

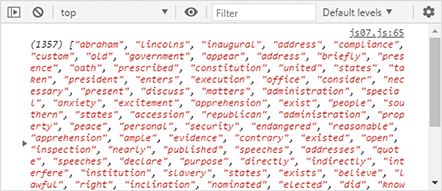
1. Return to the **js07.js** file in your code editor.
2. Directly after the for loop that removes stop words from the source text, add the following code.

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Place the remaining words in array. Line 2. let words, equals, source Text, dot, split, left parenthesis, forward slash, backward slash, s, plus, forward slash, g, right parenthesis, semi-colon.

1. Change the statement that writes to the debugger console from console.log(sourceText) to **console.log(words)**. See Figure 13.

  
**Figure 13. Splitting a text string to an array**

1. Save your changes to the file and then reload **js07.html** in your browser and load **lincoln1.html** in the app.
2. View the console log to examine the contents of the words array (see Figure 14).

  
  
**Figure 14. Contents of the words array**

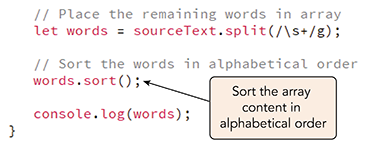
After removing the stop words, there are 1357 words left in the speech, many of which are duplicates.

**Part 9. Sort the contents of the words array:**

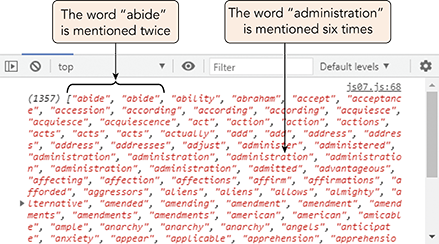
Use the sort() method to sort the contents of the words array in alphabetical order.

1. Return to the **js07.js** file in your code editor and go to the wordCloud() function.
2. Directly after the statement to create the words array, add the following statement to sort the contents of that array (see Figure 15).

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Sort the words in alphabetical order. Line 2. words, dot, sort, left parenthesis, right parenthesis, semi-colon.

  
  
**Figure 15. Applying the sort() method**

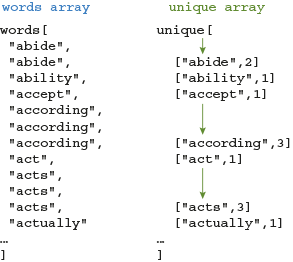
1. Save your changes to the file and then reload **js07.html** in your browser and load **lincoln1.html** in the app.
2. Use the debugger console to verify that the contents of the array are sorted in alphabetical order (see Figure 16).

  
  
**Figure 16. The words array sorted in alphabetical order**

**Part 10. Sort the contents of the words array:**

With the words array sorted, it is easier to see the duplicates. For example, the word “abide” appears twice in Lincoln’s speech, “according” appears three times, and “administration” appears six times. You will store this information in a multidimensional array named unique in which each item in the array is itself an array consisting of two entries: the first containing the text of the word and the second storing the number of times that word was used.

Figure 17 shows a preview of this “array of arrays” for a selection of words.

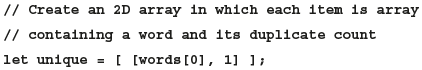
  
  
**Figure 17. The unique multidimensional array**

To create the unique array, use a for loop that iterates through the items in the words array. If the current word is different from the previous word, add it to the unique array and set the duplicate count to 1, but if it is the same as the previous word, increase the duplicate count by 1 without adding a new array item.

The name of each word in this multidimensional array can be referenced with the expression unique[*i*][0] where *i* is the index of the word in the unique array. The duplicate count is referenced with the expression unique[*i*][1]. For example, the first word “abide” from Figure 17 referenced as unique[0][0] and its duplicate count is referenced as unique[0][1]. The seventh word “actually” is referenced with the expression unique[6][0] and its duplicate count with unique[6][1].

Add code to the wordCloud() function to generate the unique array.

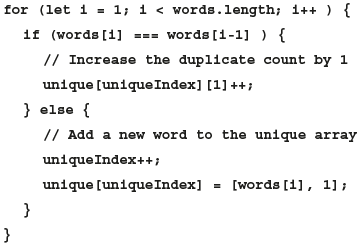
1. Return to the **js07.js** file in your code editor.
2. Directly after the statement to sort the words array, add the following statements to declare the unique array and enter its initial word and set that duplicate count to 1.



1. As you generate the unique array, you will need to keep track of the index of the current item in the array. Add the following code to declare that variable:

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Keep an index of the unique words. Line 2. let unique Index, equals, 0, semi-colon.

1. Add the following for loop to iterate through each item in the words array, adding new words as they are found or increasing the duplicate count for words previously discovered.



1. Delete the console.log(words); statement because you will no longer be needing it.

Figure 18 describes the newly added code in the file.

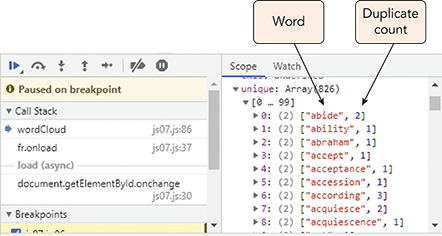
A code block with code to generate the array of unique words. Program code. In the code, the words in the variable names are merged, and the code contains the following keywords: let, for, if, else. Line 1, indented twice: Forward slash, forward slash, Sort the words in alphabetical order. Line 2, indented twice: words, dot, sort, left parenthesis, right parenthesis, semicolon. Line 3: Blank. Line 4, indented twice: Forward slash, forward slash, Create a 2 D array in which each item is an array. Line 5, indented twice: Forward slash, forward slash, containing a word and its duplicate count. Line 6, indented twice: let, unique, equals, left bracket, left bracket, words, left bracket, 0, right bracket, comma, 1, right bracket, right bracket, semicolon. Line 7: Blank. Line 8, indented twice: Forward slash, forward slash, Keep an index of the unique words. Line 9, indented twice: let, unique Index, equals, 0, semicolon. Line 10: Blank. Line 11, indented once: for, left parenthesis, let, i, equals, 1, semicolon, i, less than, words, dot, length, semicolon, i, plus, plus, right parenthesis, left bracket. Line 12, indented 3 times: if, left parenthesis, words, left bracket, i, right bracket, equals, equals, equals, words, left bracket, i, minus, 1, right bracket, right parenthesis, left brace. Line 13, indented a few times: Forward slash, forward slash, Increase the duplicate count by 1. Line 14, indented a few times: unique, left bracket, unique Index, right bracket, left bracket, 1, right bracket, plus, plus, semicolon. Line 15, indented 3 times: Right brace, else, left brace. Line 16, indented a few times: Forward slash, forward slash, Add a new word to the unique array. Line 17, indented a few times: unique Index, plus, plus, semicolon. Line 18, indented a few times: unique, left bracket, unique Index, right bracket, equals, left bracket, words, left bracket, i, right bracket, comma, 1, right bracket, semicolon. Line 19, indented 3 times: Right brace. Line 20, indented twice: Right brace. Line 21, indented once: Right brace. Line 22: Right brace, semicolon. In line 6 of the above code, the first element in the array contains the first word and a word count of 1. Line 9 of the code keeps count of the number of unique words. Line 12 tests whether the current word equals the previous word. If it does, line 14 increases its duplicate count by 1. Otherwise, a new entry is added to the unique array in lines 17 and 18.

**Figure 18. Code to generate the array of unique words**

**Part 11. View the contents of the unique array:**

Verify that the unique array has been properly constructed by viewing its contents in the Scope window of your browser’s debugger.

1. Save your changes to the file and then reload **js07.html** in your browser.
2. In your browser debugger set a breakpoint at the last line of the wordCloud() function directly before the final line of the anonymous function for the onload event hander.
3. Click the **Choose File** button and open the **lincoln1.html** file.
4. Go to the Scope window in your browser debugger and examine the contents of the unique array as shown in Figure 19.



**Figure 19. Contents of the unique array**

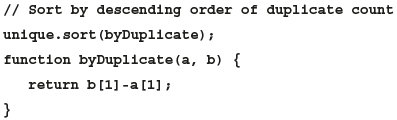
1. Continue executing the script and then remove the breakpoint from the debugger.

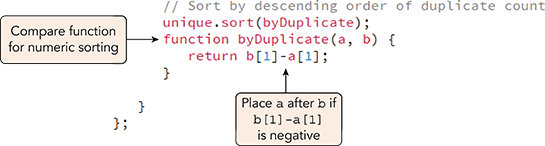
The words in the unique array are sorted in alphabetical order but it would be more useful in building the word cloud if the most often-used words were listed first.

**Part 12. Sort the unique array by duplicate count:**

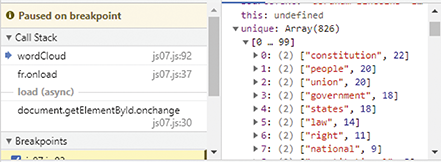
Add this compare function to the Word Cloud app and sort the words in the unique array.

1. Return to the **js07.js** file in your code editor and add the following code to the wordCloud() function as described in Figure 20:



  
  
  
**Figure 20. Using sort() with a compare function**

1. Save your changes to the file and then reload **js07.html** in your web browser.
2. In your browser debugger set a breakpoint at the last line of the wordCloud() function.
3. Click the **Choose File** button and open the **lincoln1.html** file.
4. Go to the Scope window in your browser debugger. Figure 21 shows the sorted contents of the unique array.

  
  
**Figure 21. Content of the unique array sorted by descending order of duplicate count**

Based on your analysis, “constitution” is most-repeated word in the Lincoln speech with 22 mentions, followed by “people”, “union”, “government”, and “states”.

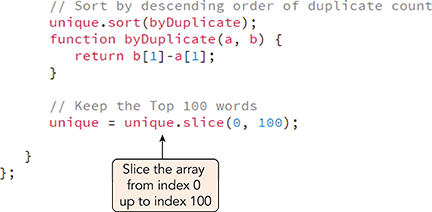
1. Continue executing the script and then remove the breakpoint from the debugger.

There are 826 unique words in the Lincoln speech, which is too many for a word cloud. You will reduce the array to the 100 most-repeated words.

**Part 13: Apply the splice() method:**

Use the slice() method to retain the first 100 words in the unique array, removing the rest.

1. Return to the **js07.js** file in your code editor and add the following code to the wordCloud() function as described in Figure 22.

Program code. In the code, the words in the variable names are merged. Line 1. forward slash, forward slash, Keep the Top 100 words. Line 2. unique, equals, unique, dot, slice, left parenthesis, 0, comma, 100, right parenthesis, semi-colon.

**Figure 22. Slicing an array**

1. Save your changes to the file and then reload **js07.html** in your web browser.
2. In your browser debugger set a breakpoint at the last line of the wordCloud() function.
3. Click the **Choose File** button and open the **lincoln1.html** file.
4. Go to the Scope window in your browser debugger and verify that the size of the unique array is reduced to the 100 most-used words.
5. Continue executing the script and then remove the breakpoint from the debugger.
6. Close the browser debugger.

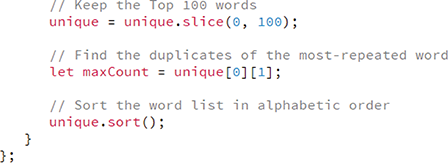
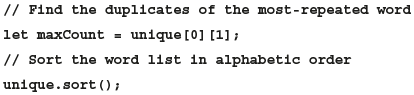
**Part 14. Store the maximum duplicate count:**

The last part of the Word Cloud app is to display the top 100 words with the font size of each word proportional to the number of times it was repeated. The word with the most duplicates will have the largest size; the word with the fewest duplicates will have the smallest. To size these words correctly, the app will need the count of the word with the most duplicates. Because the unique array has been sorted in descending order of duplicate count, the first item in the array will also be the most repeated and, thus, the top count can be stored using the following expression:

Program code. In the code, the words in the variable names are merged. Line 1. let max Count, equals, unique, left bracket, 0, right bracket, left bracket, 1, right bracket, semi-colon.

Apply this command to the wordCloud() function and then sort the unique array back into alphabetical order.

1. Return to the **js07.js** file in your code editor.
2. Directly below the command that slices the unique array, add the following commands to the function as shown in Figure 23.



**Figure 23. Determine the count of the most-repeated words**

**Part 15. Display each word in a size proportional to its use:**

To display the word list sized proportional to each word’s use, create a for loop that iterates through every word in the unique array. For each word, set its size in em units as a fraction of the size of the word with the most repetitions. The most-used word will be displayed with a font size of 1em. A word that appears half as often will have a font size to 0.5em. A word that appears a tenth as often has a font size of 0.1em, and so forth. The font size represented by 1em is set in the CSS style sheet used with this project and could be changed by the web designer based on the website design without affecting the script code.

1. Return to the **js07.js** file in your code editor.
2. Directly below the command to sort the unique array, add the following commands to reference the element that will contain the word cloud and set its initial content to an empty string.



1. Add the following for loop to add each word to the cloudBox, sized proportional to the number of times it appears in the document.

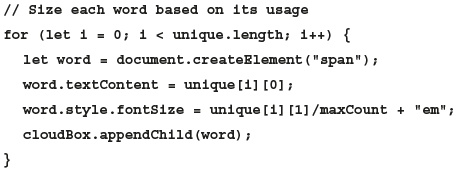


Figure 24 describes the code in the file.

A code block with code for sizing the words in the word cloud. Program code. In the code, the words in the variable names are merged, and the code contains the following keywords: let, for. Line 1, indented twice: Forward slash, forward slash, Sort the word list in alphabetic order. Line 2, indented twice: unique, dot, sort, left parenthesis, right parenthesis, semicolon. Line 3: Blank. Line 4, indented twice: Forward slash, forward slash, Reference the word cloud box. Line 5, indented twice: let, cloud Box, equals, document, dot, get Element By I d, left parenthesis, left double quotation mark, w c underscore cloud, right double quotation mark, right parenthesis, semicolon. Line 6, indented twice: cloud Box, dot, inner H T M L, equals, left double quotation mark, right double quotation mark, semicolon. Line 7: Blank. Line 8, indented twice: Forward slash, forward slash, Size each word based on its usage. Line 9, indented twice: for, left parenthesis, let, i, equals 0, semicolon, i, less than, unique, dot, length, semicolon, i, plus, plus, right parenthesis, left brace. Line 10, indented 3 times: let, word, equals, document, dot, create Element, left parenthesis, left double quotation mark, span, right double quotation mark, right parenthesis, semicolon. Line 11, indented 3 times: word, dot, text Content, equals, unique, left bracket, i, right bracket, left bracket, 0, right bracket, semicolon. Line 12, indented 3 times: word, style, font Size, equals, unique, left bracket, i, right bracket, left bracket, 1, right bracket, forward slash, max Count, plus, left double quotation mark, e m, right double quotation mark, semicolon. Line 13, indented 3 times: cloud Box, dot, append Child, left parenthesis, word, right parenthesis, semicolon. Line 14, indented twice: Right brace. Line 15, indented once: Right brace. Line 16: Right brace, semicolon. Line 9 contains code that loops through every word in the list. Lines 10 and 11 create a span element containing the text of every word. Line 12 sets the font size as a percentage of the largest duplicate count. Line 13, appends the word to the word cloud box.

**Figure 24. Sizing the words in the word cloud**

1. Close the js07.js file, saving your changes.
2. Reload the **js07.html** file in your browser.
3. Click the **Choose File** button and load the **lincoln1.html** file. A word cloud is generated for the content of Lincoln’s speech shown in your reading.
4. Click the **Choose File** button again and load the **lincoln2.html** speech for Lincoln’s second inaugural address given near the end of the Civil War (see Figure 25).



**Figure 25. Word cloud for Lincoln’s second inaugural address**

You can compare the word cloud in Figure 25 with the one shown earlier in your reading, to see how the tone and emphasis of Lincoln’s speech changed between the First and Second Inaugural addresses.

1. Use the Word Cloud app to open the **fdr3.html**, **jkf1.html**, and **reagan1.html** files for those presidential inaugural addresses. What do the word clouds tell you about the content and theme of those speeches?
2. Close the js07.html file.

You have completed your work on the Word Cloud app. The app can be used with almost any text file to generate a word cloud highlighting the most important themes and concepts. By modifying the style sheet, you can create a wide variety of word cloud designs and styles.