Word Cloud

Making a Word Frequency Map Object

Word Frequency Map for a Word Cloud

- A Word Cloud is a visual representation of the frequency of words in a string, such as found in a blog post or other article.
- In a word cloud, the most frequently occurring words appear in the largest font size in order to do this, the frequency of words must be known.
- Therefore, before we can make a Word Cloud, we have to make what is called a Word Frequency Map from of the words. This takes the form of an Object, where the keys are unique words and the value of each key is the number of times the word occurs.
- To make the Word Frequency Map, we have to convert the text to an array, with each item a word.
- Then we loop through the array of words. every time a unique word is found, the object is assigned that word as a new key with an initial value of 1.
- The next time the word is encounterd, no new key is made, but rather the value of the existing key is incremented by 1.
- The resulting Word Frequency Map can then be used to make a Word Cloud by setting the font size of each word based on the frequency, with most frequent words biggest also
- A Word Cloud only contains interesting keywords, so there needs to be a filter that prevents what are known as **stopwords** from being included in the Word Frequency Map. Stopwords include such common words as **'the', 'and', 'of', 'on', 'with'**, etc.

1. Open **stopwords.js** and have a look at the array of stopwords. These are the words that must be excluded from our Word Cloud.

2. Define a function called **makeWordFreqMap** with two parameters: **str** and **stopwords**. The **str** will be an entire passage, such as an article, story or blog post.

```
function makeWordFreqMap(str, stopwords) {
}
```

3. Call the **trim()** method on the string. This removes all empty, extra spaces. Also chain on **toLowerCase()** so that "Land" and "land" are not regarded as different words:

```
function makeWordFreqMap(str, stopwords) {
    str = str.trim().toLowerCase();
```

```
    }
```

4. Remove punctuation so that "world." and "world" are not saved as separate keys. You can individually target punctuation markes, or, better, use **RegEx** (Regular Expressions):

```
        // str = str.replace(",", "");
        // str = str.replace(".", "");
        // str = str.replace(";", "");
        // str = str.replace(":", "");
        // str = str.replace("?", "");
        // str = str.replace("!", "");
        // or use fancy RegEx move to strip all non-alphanumeric
characters
        str = str.replace(/[^\w\s]|_/g, "").replace(/\s+/g, "
").replace(/[0-9]/g, '');
```

- **RegEx gibberish decoded**
- **\w** is any digit, letter, or underscore.
- **\s** is any whitespace.
- **[^\w\s]** is anything that's not a digit, letter, whitespace, or underscore.
- **[^\w\s]|_** is the same as #3 except with the underscores added back in.
- **[0-9]** is all digits
- **/g** is globally replace (everywhere)
- **str = str.trim()** get rid of any extra whitespace that still may remain

5. Use the **split()** method to make an array from all the words in **str**:

```
        let arr = str.split(" ");
```

6. Declare a new object. This is for storing key-value pairs of word frequencies:

```
        let obj = {};
```

7. Loop through the array of words made from the big string:

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

8. Inside the loop, save the current array item as **word**:

```
            let word = arr[i];
```

9. Use conditional logic to see if the current word is NOT included in the stopwords array:

```
            if(!stopwords.includes(word)) { // if the current
             }
```

10. If the current word is not already an object key, make it a key, with an initial value of 1; else increment the value of the word key by 1. A ternary is ideal for this. This ends the loop:

```
                !obj[word] ? obj[word] = 1 : obj[word]++;

            } // end if-else
        } // end for loop
```

11. Return the object. This is the Word Cloud / Word Frequency Map. This ends the function

```
            return obj; // output the function
    } // end function
```

12. Call the function twice, each time passing in a separate story, imported into the html page:

```
    let fairyTaleWordFreq = makeWordFreqMap(textPassage, stopwords);
    console.log('fairyTaleWordFreq:', fairyTaleWordFreq);

    let treehouseWordFreq = makeWordFreqMap(treehouse, stopwords);
    console.log('treehouseWordFreq:', treehouseWordFreq);
```

END: Lesson 05.08
NEXT: Lesson 06.01