Summer 2019 - Writing Exercise - Ryan Fleck

Thank you for your interest in Me! I have received your request to complete writing exercises, and have elected to complete exercises **2** and **4**.

As a Co-Op developer, I've spent a lot of time documenting systems for non-technical users and system administrators. Having enjoyed that part of the work, I decided to apply for this technical writing internship. With any luck, you'll deem my responses eloquent, detailed, and brief enough to be considered for an interview; see you then!

Table of Contents

Ex.2: Add a Bar Chart to a Google Spreadsheet **2**

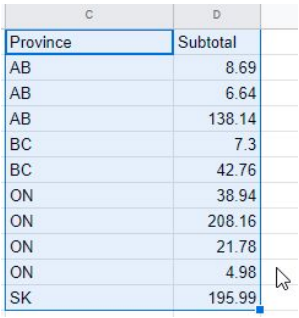
Ex.4: Code Review **3**

Implementation Breakdown 3

Suggestions for Improvement 4

Ex.2: Add a Bar Chart to a Google Spreadsheet

Bar charts are often used for comparing sets of metrics between groups. To use a bar chart with your data, first select the cells you would like to display, including column headers.

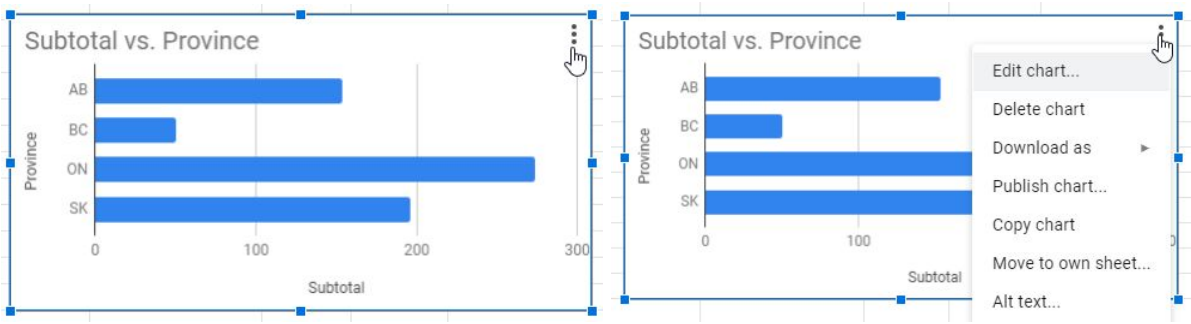


Province	Subtotal
AB	8.69
AB	6.64
AB	138.14
BC	7.3
BC	42.76
ON	38.94
ON	208.16
ON	21.78
ON	4.98
SK	195.99

Next, click **insert chart** from the ribbon. You may have to use the **more** button to expose the *formulas and filters* section if your window is narrow.

The *Chart editor* will now be displayed. Navigate to the *Chart type* dropdown, click, and select one of *Bar chart, Stacked bar chart,* or *100% stacked bar chart*.

A new *Bar chart* will now be placed on the page. The display of data can now be tweaked with the options available in the *Chart editor.* If additional modifications need to be made, this menu can be reopened by selecting the menu indicator present in the top-right of the bar chart, and clicking **Edit chart…** . Double-clicking on any label will open a text field for modification.



Ex.4: Code Review

The following Java code was submitted for feedback on January 28, 2019:

```
public static void findNeedles(String haystack, String[] needles) {
if (needles.length > 5) {
      System.err.println("Too many words!");
}

else {
      int[] countArray = new int[needles.length];
      for (int i = 0; i < needles.length; i++) {
            String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);
            for (int j = 0; j < words.length; j++){
                  if (words[j].compareTo(needles[i]) == 0) {
                        countArray[i]++;
                  }
            }
      }
      for (int j = 0; j < needles.length; j++) {
            System.out.println(needles[j] + ": " + countArray[j]);
      }
}
}
```

Both an *Implementation Breakdown* and *Suggestions for Improvement* have been provided in response to this request.

Implementation Breakdown

Comments are provided below the relevant code snippets.

```
if (needles.length > 5) {
      System.err.println("Too many words!");
}
```

A simple test to ensure that fewer than six strings (‘needles’) are being passed to the main method. It’s good that this is printed to **stderr** rather than **stdout**. The remainder of method functionality is contained in the *else* case for this *if* statement.

```
int[] countArray = new int[needles.length];
```

An array of the same length as the ‘needles’ array is initialized to keep track of the number of times each *needle* appears in the *haystack.* As the method runs, this array is incremented whenever a needle at array address X **needs work**

```
for (int i = 0; i < needles.length; i++) {
      String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);
      for (int j = 0; j < words.length; j++){
            if (words[j].compareTo(needles[i]) == 0) {
                  countArray[i]++;
            }
      }
}
```

This $O(n^2)$ algorithm begins by executing a for loop for each element in the *needles* array. For each *needle*, the haystack is split into a String array of *words.*

The algorithm continues by executing a nested for loop, iterating over each element in the *words* array. For each *word*, a comparison is made with the current *needle* string as selected by the parent for loop. If a needle matches a word, the index of *needle* is used to increment *countarray* at the index responsible for tracking the occurences of each *needle*.

```
for (int j = 0; j < needles.length; j++) {
      System.out.println(needles[j] + ": " + countArray[j]);
}
```

Finally, the last *for* loop simultaneously runs through the arrays *needle* and *countArray* to print the occurences of each *needle* in the string *haystack*. The method is finished, and exits.

Suggestions for Improvement

Two suggestions are provided:

1. Array *words* should be moved outside of the first for loop.
2. A more efficient method could be used to replace compareTo.

Array *words* should be moved outside of the first for loop.

```
for (int i = 0; i < needles.length; i++) {
      String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);
      for (int j = 0; j < words.length; j++){
            if (words[j].compareTo(needles[i]) == 0) {
                  countArray[i]++;
            }
      }
}
```

As the array *words* is not modified while being compared to a single *needle*, it is recommended to move this array initialization outside the nested for loops. This will prevent the copy and split operations from running once for each *needle*.

A more efficient method could be used to replace compareTo.

```
for (int i = 0; i < needles.length; i++) {
      String[] words = haystack.split("[ \"\'\t\n\b\f\r]", 0);
      for (int j = 0; j < words.length; j++){
            if (words[j].compareTo(needles[i]) == 0) {
                  countArray[i]++;
            }
      }
}
```

The compareTo() method compares two strings to determine lexicographical equality. More processing than is necessary is done for a simple string comparison. The method equals() is recommended to improve the runtime of this algorithm.