Eric Roberts and Jerry Cain
CS 106J

Handout #32S
May 10, 2017

Solution to Section #5

Portions of this handout by Eric Roberts

Helper functions useful for all problems

```
/*
 * Simple helper function that creates an array of size n
 * where each element is initialized to value.
 */
function createArray(n, value) {
   var array = [];
   for (var i = 0; i < n; i++) {
      array.push(value);
   }
   return array;
}

/*
 * Helper function that takes a string and creates an array
 * with the same size of the alphabet. Index 0 represents 'a',
 * index 1 represents 'b', etc. Each array cell keeps
 * track of how many times that letter occurs in the string str.
 */
function countAlphabetFrequencies(str) {
   var alphabet_counts = createArray(alphabet.length, 0);
   var base = "a".charCodeAt(0);
   for (var i = 0; i < str.length; i++) {
      alphabet_counts[str[i].charCodeAt(0) - base]++;
   }
   return alphabet_counts;
}
```

1. Find duplicate characters

```
/*
 * Takes a string str and returns all the letters
 * that are duplicates/have a frequency greater than 1. The returned
 * format is "letter: count; letter: count; ...".
 */
function findDupChars(str) {
   str = str.toLowerCase();
   var base = "a".charCodeAt(0);
   var alphabet_counts = countAlphabetFrequencies(str);
   var result = "";
   for (var i = 0; i < alphabet.length; i++) {
      if (alphabet_counts[i] >= 2) {
         var letter = String.fromCharCode(base + i);
         result += letter + ": " + alphabet_counts[i] + "; ";
         //to use as a removeDupChars helper function, use the
         //below definition of result instead of the one above
         //result += letter;
      }
   }
   return result;
}
```

2. Remove duplicate characters

```
/*
 * Takes a string str, finds all the letters
 * that are duplicates/have a frequency greater than 1, and
 * returns a new string that has all the duplicate letters
 * removed from the original string str.
 */
function removeDupChars(str) {
   var dupChars = findDupChars(str);
   var result = "";
   for (var i = 0; i < str.length; i++) {
      if (dupChars.indexOf(str[i]) === -1) {
         result += str[i];
      }
   }
   return result;
}
```

3. Is anagram

```
/*
 * Takes two strings str1 and str2, and if the two strings
 * are valid anagrams, the function returns true. Otherwise,
 * the function returns false. An anagram is defined as a word
 * that can be formed by rearranging the letters of another word.
 */
function isAnagram(str1, str2) {
   var counts1 = countAlphabetFrequencies(str1);
   var counts2 = countAlphabetFrequencies(str2);
   for (var i = 0; i < counts1.length; i++) {
      if (counts1[i] !== counts2[i]) return false;
   }
   return true;
}
```

4. First non-repeating character

```
/*
 * Takes a string str and returns the first letter
 * in str that is unique.
 */
function firstNonRepeatingChar(str) {
   var counts = countAlphabetFrequencies(str);
   var base = "a".charCodeAt(0);
   for (var i = 0; i < str.length; i++) {
      var letter_index = str[i].charCodeAt(0) - base;
      if (counts[letter_index] === 1) {
         return str[i];
      }
   }
   return "No valid character exists.";
}
```

5. indexOf

```
/*
 * An implementation of indexOf, which takes a string str and
 * a pattern to match. This returns -1 if pattern does not exist
 * in the string str. If pattern does exist, this returns the
 * index in str where the first instance of pattern begins.
 */
function indexOf(str, pattern) {
   var numMatched = 0;
   for (var i = 0; i < str.length; i++) {
      if (str[i] === pattern[numMatched]) {
         numMatched++;
         if (numMatched === pattern.length) {
            return i - numMatched + 1;
         }
      } else {
         numMatched = 0;
      }
   }
   return -1;
}
```

6. Remove a specific word from string

```
/*
 * Takes a string manuscript of arbitrary length and removes every
 * instance of toRemove from the manuscript. Then this retuns the
 * cleansed manuscript.
 */
function removeUnwantedWord(manuscript, toRemove) {
   var nextIndexToRemove = manuscript.indexOf(toRemove);
   while (nextIndexToRemove !== -1) {
      manuscript = manuscript.substring(0, nextIndexToRemove)
                   + manuscript.substring(nextIndexToRemove
                   + toRemove.length);
      nextIndexToRemove = manuscript.indexOf(toRemove);
   }
   return manuscript;
}
```