**Message Mixer**

Message Mixer is a messaging service that allows you to perform an action on input text and display the output of that behavior to the console. For example, with the current functions defined in Message Mixer, you can:

* count the characters in a message
* capitalize the first character of words
* reverse a message’s words in place
* reversing characters in place
* replace the first occurrence of a string
* replace all occurrences of a string
* encode text by swapping certain characters for other characters

At present, Message Mixer runs as a program in a single file. This single file includes functions that define behavior as well as the output. Message Mixer knows that by extracting the functions into a module, logic can be reused in different parts of our application.

Let’s help Message Maker turn the program into a module!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**1/17Complete**

Mark the tasks as complete by checking them off

**1.**

In the code editor, you have a file **messageMixer.js**. Run the file to see the output of the program.

Hint

Save your code to see the output.

**2.**

Each of the six functions within **messageMixer.js** manipulates a string of text in some way, and the displayMessage() function outputs a manipulated message to the console.

Segment the text-transformation behavior that should be kept within the module **messageMixer.js**, and the output behavior that should exist within **message.js**.

Copy the displayMessage() function and the displayMessage() function call and paste it in **message.js**. Then delete the displayMessage() function from **messageMixer.js**.

We will come back to this function soon. Notice that if you run the file, you will see a ReferenceError.

Hint

Place this code in **message.js**, removing it from **messageMixer.js**.

function displayMessage() {

// ...

}

displayMessage();

**3.**

You can now turn the behavior of **messageMixer.js** into a module. In this file, create a variable MessageMixer and set it equal to an empty object to represent the module as an object.

Hint

const MessageMixer = {};

**4.**

You’ve defined a MessageMixer object, but in order for you to attach behavior to it, you’ll need to make sure that each function is accessible as a property of it.

One by one, for each function in **messageMixer.js**, modify the function so that it is a property on the object.

Hint

Each function should be a property on the MessageMixer object, like this:

MessageMixer.countCharacter = function(inputString, inputCharacter) {

let count = 0;

let string = inputString.toLowerCase();

let character = inputCharacter.toLowerCase();

for (let i = 0; i < string.length; i++) {

if (string[i] === character) {

count++;

}

}

return count;

}

**5.**

In addition to to setting each function as a property on the MessageMixer object, notice that several functions depend on another function to be called.

The reverseAllWords() function, for instance, calls the reverseWord() in its function body. In order for this function to work, reverseWord() must be prepended by MessageMixer..

The encode() function works in a similar way, when it calls replaceAllOccurrences(). Both need to be prepended by the module name.

Refactor the reverseAllWords() and encode() functions so they use MessageMixer on function calls within the function.

Hint

MessageMixer.reverseAllWords = function(sentence) {

let words = sentence.split(' ');

for (let i = 0; i < words.length; i++) {

words[i] = MessageMixer.reverseWord(words[i]);

}

return words.join(' ');

};

...

MessageMixer.encode = function(string) {

const replacementObject = {

a: '@',

s: '$',

i: '!',

o: '0'

};

for (let key in replacementObject) {

string = MessageMixer.replaceAllOccurrences(string, key, replacementObject[key]);

}

return string;

};

**6.**

Your MessageMixer object now has properties. Export it using module.exports syntax.

Hint

module.exports = MessageMixer;

**7.**

Import the module in **message.js** using the require() statement. You can use the variable MessageMixer again to represent the module.

When you run **message.js**, you will still get a ReferenceError.

Hint

const MessageMixer = require('./messageMixer.js');

**8.**

In order for **message.js** to display the output from the module, you’ll need to call each of the functions in the displayMessage() function as properties of the imported MessageMixer object.

When you run the file, you should see the output of the functions.

Hint

Each of the function calls in **message.js** should be a property of the MessageMixer module.

function displayMessage() {

console.log(MessageMixer.countCharacter('My name is Bana', 'A'));

// ...

}

**9.**

Continue by adding a few additional functions to the MessageMixer module.

In **messageMixer.js**, create a function on the MessageMixer object called palindrome() that takes a String str as a parameter.

The body of the function should use string concatenation or interpolation to return the string, a space, and the reverse of the string. You can use the reverseWord() function. You will need to call reverseWord() as a method of MessageMixer.

Hint

MessageMixer.palindrome = function(str){

return `${str} ${MessageMixer.reverseWord(str)}`;

};

**10.**

Again, in **messageMixer.js**, create another function pigLatin() on the MessageMixer object that takes a sentence and a character as parameters.

The body of the function should return the sentence split at each of the spaces, and joined back together using the character argument concatenated with a ' '.

Hint

MessageMixer.pigLatin = function(sentence, character) {

return sentence.split(' ').join(character + ' ');

};

**11.**

In **messageMixer.js**, use modify the way you export MessageMixer to use export default instead of. module.exports.

Hint

export default MessageMixer;

**12.**

In **message.js**, use the import keyword to import the MessageMixer module. The file path will be './messageMixer'.

Hint

import MessageMixer from './messageMixer';

**13.**

In **message.js**, use console.log() to display the output of palindrome() and pigLatin() functions within the displayMessage() function. You will need to pass the functions a string.

You should see the output of the manipulated message.

Hint

function displayMessage() {

// ...

console.log(MessageMixer.pigLatin("What is the color of the sky?", "ay "));

console.log(MessageMixer.palindrome("What is the color of the sky?"));

}

displayMessage();

or

const displayMessage = () => {

// ...

console.log(MessageMixer.pigLatin("What is the color of the sky?", "ay "));

console.log(MessageMixer.palindrome("What is the color of the sky?"));

}

displayMessage();

**14.**

Message Mixer wants to test one final behavior for their program. They’d like to export each of the functions as a variable. Modify **messageMixer.js** so that each function is exported as a variable name.

Note that as you do this, you should also remove the MessageMixer object in front of each function, as well at any reference to MessageMixer in the body of the reverseAllWords(), encode(), and palindrome() functions.

Hint

For example, instead of

MethodMixer.reverseWord = function (word) {

// ...

};

You would write:

function reverseWord(word) {

// ...

};

or

const reverseWord = (word) => {

// ...

}

or

const reverseWord = function(word) = {

// ...

}

**15.**

Using the export statement at the bottom of the file, export each of the function by their variable named between two {}

Hint

export { countCharacter, capitalizeFirstCharacterOfWords, reverseWord, reverseAllWords, replaceFirstOccurence, replaceAllOccurrences, encode, palindrome, pigLatin };

**16.**

In **message.js**, modify the program to import each of the variables.

Hint

import { countCharacter, capitalizeFirstCharacterOfWords, reverseWord, reverseAllWords, replaceFirstOccurence, replaceAllOccurrences, encode, palindrome, pigLatin} from './messageMixer';

**17.**

As a last and final step, modify each of the functions within each of the displayMessage() function so that they use only the variable name in the function call.

Hint

function displayMessage() {

// ...

console.log(pigLatin("What is the color of the sky?", "ay "));

console.log(palindrome("What is the color of the sky?"));

}

displayMessage();