Assignment 4: Vigenere Cipher

What you need to do

- Build a client-side web application using JavaScript
- Execute JavaScript logic based on user-generated events within the DOM

Requirements

In this assignment, you will implement the *Vigenere cipher* algorithm in JavaScript and provide an HTML user interface to allow a user to enter a plain-text message and a key that will be used to generate ciphertext. Additionally, the cipher-text can be decrypted using the same key value. For information and algorithm details regarding the Vigenere cipher, please see this link: Vigenere Cipher (Wikipedia)

vigenère cipher

You must implement both encryption and decryption logic within the provided JavaScript file (*vigenere.js*). The archive file that contains the skeleton JavaScript file as well as some other supporting code files is provided with the assignment at the D2L course site. The function's signatures for encrypt and decrypt have been defined for you in this file. Do not change the function signatures, but feel free to add additional functions to help organize your code. The automated unit test cases call the encrypt/decrypt methods directly. In addition, do not rename *vigenere.js*.

The automated unit test cases are based on QUnit, which is a JavaScript unit testing framework. QUnit test code is given in *tests.js* in the provided code. You should not change *Tests.html* and *tests.js* in the provided code package. You can use the displayed output at the right part on your index.html to help you develop your *vigenere.js*.

The following constraints apply to your implementation of the Vigenere Cipher:

- Ignore spaces. If a space is encountered in the plaintext/ciphertext, the resulting string should include a space at the same location, and your code should continue to process the remainder of the input string. For example, based on the example in Vigenere Cipher (Wikipedia), your implementation should be able to succeed the following encryptions:

Example 1:

Plaintext:	ATTACKATDAWN
Key:	LEMON
Ciphertext:	LXFOPVEFRNHR

Example 2:

Plaintext:	ATTACKAT DAWN
Key:	LEMON
Ciphertext:	LXFOPVEF RNHR

- Ignore spaces in the key. If a space is encountered in key, your code should remove the space and concatenate the letters in the key to process the input string.

Example 3:

Plaintext:	ATTACKATDAWN
Key:	LE M ON
Ciphertext:	LXFOPVEFRNHR

Example 4:

Plaintext:	ATTACKAT DAWN
Key:	LE M ON
Ciphertext:	LXFOPVEF RNHR

- Maintain case between the plaintext/ciphertext messages. If a character at a position is upper- case, the substituted letter should also be upper-case in the resulting string.

- Ignore special characters such as punctuation of a plaintext/ciphertext message. If you encounter punctuation in the message, treat it similarly to whitespace. For example: if a comma (,) appears in the input string, it should also appear in the same position of the output string.
- You must throw a JavaScript exception if you receive invalid input to the encrypt/decrypt functions. This would include null values or empty strings or non-alphabetic characters within the key. Spaces can appear in the key, but numbers or other special characters should not. For example, below I have some sample code when defining function encrypt(plaintext, key). The plaintext is to be encrypted based on the key by calling the function.

```
function encrypt(plaintext, key)
{
  //some code prior to the invalid input handling
  //throw an exception when the plaintext is empty string
  If (plaintext === “”)
  {
     throw false;
  }
  else if (some condition to check if the key is invalid)
  {
     throw false;
  }
  //other code
}
```

Without throwing the exceptions, the QUnit test code (Tests.html and tests.js) cannot test your code appropriately.

For example, in ***test.js***, lines 57-68 are used to test if the encrypt(plaintext, key) handles empty plaintext input appropriately.

```
QUnit.test('Encryption - empty string for plaintext', function (assert) {
   var plaintext = "";
   var key = "key";
   var thrownException = null;
   var ciphertext = null;
   try {
      ciphertext = encrypt(plaintext, key);
   } catch (err) {
      thrownException = err;
   }
   assert.notOk(thrownException == null, "Failing because no exception is thrown on empty string for plaintext input.");
```

```
    });
```

The assert statement is used to detect if your **encrypt(plaintext, key)** function call throws an exception referred to by *err* in the code. If bold blue "throw false" is missing in the **encrypt** definition, the test case, which is the sixth one on the Homework 4 QUnit Tests section of the web page, may fail to pass and will be presented as red on the list.

HTML Form (index.html)

- You must implement HTML form controls to allow a user to enter plaintext, ciphertext, and key values.
- Form elements must be clearly labeled.
- Users must be able to invoke the encrypt/decrypt function by clicking on an HTML element, such as a button, link, image, etc. It must be obvious what the user must click on to perform each operation.
- You must display the result of the encryption/decryption operation within the web page in a way that the user can copy/paste the resulting output string.
- You must display an error message to the user indicating the error condition. Generic error messages do not count.
- You may style this page however you wish.

Implementation Tips

- Pay attention to the Description and Algebraic Description sections in Vigenere Cipher (Wikipedia). I especially like the algebraic description part, which is helpful to define the encryption/decryption algorithms in JavaScript. Be aware, however, that the algebraic description in the Wikipedia page assumes the texts including key consist only of letters. Thus, you do need to adjust the calculations by handling the special characters such as spaces/punctuations that can show up in the input strings of your algorithms.

- The other thing I want to clarify is the the description algebra described in Vigenere Cipher (Wikipedia). For the decryption logic, when (Ci - Ki) is negative, you cannot use JavaScript to calculate Mi directly based on the modulo operation described in the document. (You may try applying x % 26 operation on x that is negative and see what is the result.) Instead, you should check if (Ci - Ki) is negative. If the difference is negative, Mi should be (Ci - Ki) +26.

- Pay attention to the String methods in JavaScript. Especially the methods including *charAt(), fromCharCode(),* and *charCodeAt*().

- When you debug your JavaScript code, I strongly recommend Google Chrome Developer Tool. There is a YouTube video on how to debug using the tool included on your reading list of D2L Module JavaScript.

Turn-in

Please turn in a Zip archive where the filename is your_uanetid_h4.zip (not 7zip or some other format). This archive should contain:

index.html
vigenere.js

You also need to include the unit test files (Tests.html and tests.js) in your submission even though they should be same as the provided files. If you use an external css file to stylize your index.html, don’t forget to include the file in your package.

Please submit your zip file to the D2L dropbox for Assignment 4 by the due date and post your homework 4 files to your password-protected web space on the U-System. When submitting your zip file, please **specify the URL of your posted pages** as well as **password** for user **milazzom** to access the pages in the drop box *submission notes*.

Rubric

This assignment is worth 100 points.

80 points will be awarded based on the results of the included QUnit test cases.

You will be awarded a percentage of those points based on the number of test cases that pass, so if you pass all test cases, you will be awarded 80 points; If your code does not pass a test case, you will receive a percentage of the points based on the following equation: (passed cases/total cases) * 80.

The remaining 20 points will be assigned based on how your JavaScript works when invoked manually via the index.html file. Here is the following breakdown:

- 5 points for having an HTML form on the page.
- 5 points for having event handlers correctly configured.
- 5 points for displaying the result of the encryption/decryption routines upon the user initiating the encryption/decryption operation.
- 5 points for displaying an error message to the user if invalid plaintext, ciphertext, or key values are used.