Business Academy Aarhus

Mandatory assignment Security

URL: At Github

Students

Popoutanu Dan

Razvan Chiminit

First Exercise Example:

It all started with the exercises in class, when we had to cipher the text using a certain number. I chose the approach of working with ASCII characters.

In order to explain how I got to my final code I am first giving an easier example using 8 as shift key.

Encrypting Caesar

ASCII letters start from 65 and finish at 90 for big letters, while small case letters start from 97 up to 122. This is important because it is the main thing my code is based on.

for (var i = 0; i < n.length; i++)

var code = n[i].charCodeAt(n[i]);

I made a function using a for loop that goes over each character of the word and takes the ASCII code value. The key is very important in my example since it is only based on this that the code works.

if ((code >= 65 && code < 83) || (code >= 97 && code < 115)) {

code+=8; }

else if ((code >= 83 && code <= 90) || (code >= 115 && code <= 122)) {

code = code - 18; }

I use an if else statement to divide the alphabet in two parts, the key being the distance from where I divide calculated by subtracting the key from the alphabet (26-8 =18) . The first part starts from 65 (ASCII value of A) up until the “middle of the alphabet” OR starting from 97 (ASCII value of a) and up until the “middle of the alphabet”. If the specific character is somewhere between any of these values, the code value should change shifting 8 characters to the right.

The second part of the cipher , starts from where the previous code was left at , checking first for Upper Case letters , if the character is between the “middle of the alphabet” and the end 90 (ASCII value of Z) OR checking for lower case letters, from 122(ASCII value of z)

result+=String.fromCharCode(code);

In the beginning we had a declared variable named result that handles the final result. After everything is said and done the code value is taken back to a letter which is added to a string and thus we get to our final result.

This is the complicated function part. Other than this, we made an html form with two input fields in html, one for the key and one for the text and a div that should display the result. Data is send using the POST method and when you click on the button you activate each script, either the encrypt or the decrypt. In order to get the value the user inputs I use another function

```
var n = document.getElementById("id1").value;

var k = document.getElementById("key1").value;

return encryptCaesar(n, k);
```

This function takes the value from the two input fields that I have and saves them as a value k and n. Ran into some problems when having the same field for both the encrypt and decrypt so it seemed easier for me to make separation of concerns, that's why I made two different scripts and element for everything, keys, test, buttons, functions.

Decrypting Caesar Example

The second part of the code is very similar to the first one, same rules are applied. I have noticed that instead of adding shifting 8 to the right I could also add shifting 8 to the left by changing from plus to minus. Similar to the example above I iterate over the key word using the same for loop but I change some stuff in the if/else function :

```
if ((code >= 73 && code <= 90) || (code >= 105 && code <= 122))

{code-=8; }

else if ((code >= 65 && code < 73) || (code >= 97 && code < 105))

{ code = code + 18; }
```

This time the if else starts with "the middle of the code "to the end 90 (ASCII value of Z) OR "middle" to end 122 (ASCII value of z) and instead of adding 8 now it subtracts 8. The same goes for the second part of the code which now starts from the first half of the alphabet, from 65 to (alphabet + shifting key , which is 65 +8 = 73). Now Adding 18 instead of subtracting it.

First Exercise

Encrypting

Based on everything above I was able to get to the next logical step in my process, which is what if the user wants to choose his own key. I did not go for a dropdown menu but rather a simple input form

where the user could input anything he wants. The main challenge was finding a variable and adding it to the code so that whatever the variable is, the code still getting to the correct answer.

The main thing that I did here was adding a K element that takes the inputted value of the key. Then, based on the previous exercise I needed to find a middle point that I would subtract and add in different functions. Mainly in order to find this “middle” point I used similar things to the previous exercise, so my if statement starts from the beginning of the Upper case alphabet (65) and goes to the end of the alphabet minus the key (90 – k) || from the first lower case letter until the last minus the key. If the letter is included here the letter code gains the shift cipher value.

```
if ((code >= 65 && code <= 90 - k) || (code >= 97 && code < 123 - k)) {
    code += k ;

    //console.log(String.fromCharCode(code));
} else if ((code >= 91 - k && code <= 90) || (code >= 123 - k && code <=
122)) {
    code -= (26 - k);
```

If however it is not in the previous code, it means that the letter is in the second part of the alphabet (notice we have 91-k until 90 which would translate into middle -> end) OR (123 –k until 122) and we need to go all the way around (alphabet – key).

I used stuff in the previous exercise in order to make the code but whatever I would do it didn’t seem to work. After some struggle and debugging I figured out the problem was the k value that my input viewed as a string. I used : *k = parseInt(key) % 26* in order to make my program transform the k into a number and then added modulo 26 so whatever number you put there it will still give out a number from 1 to 26 so the code will not fail.

Decrypting

It all became more confusing when I thought about decrypting since I had to take my logic the other way around.

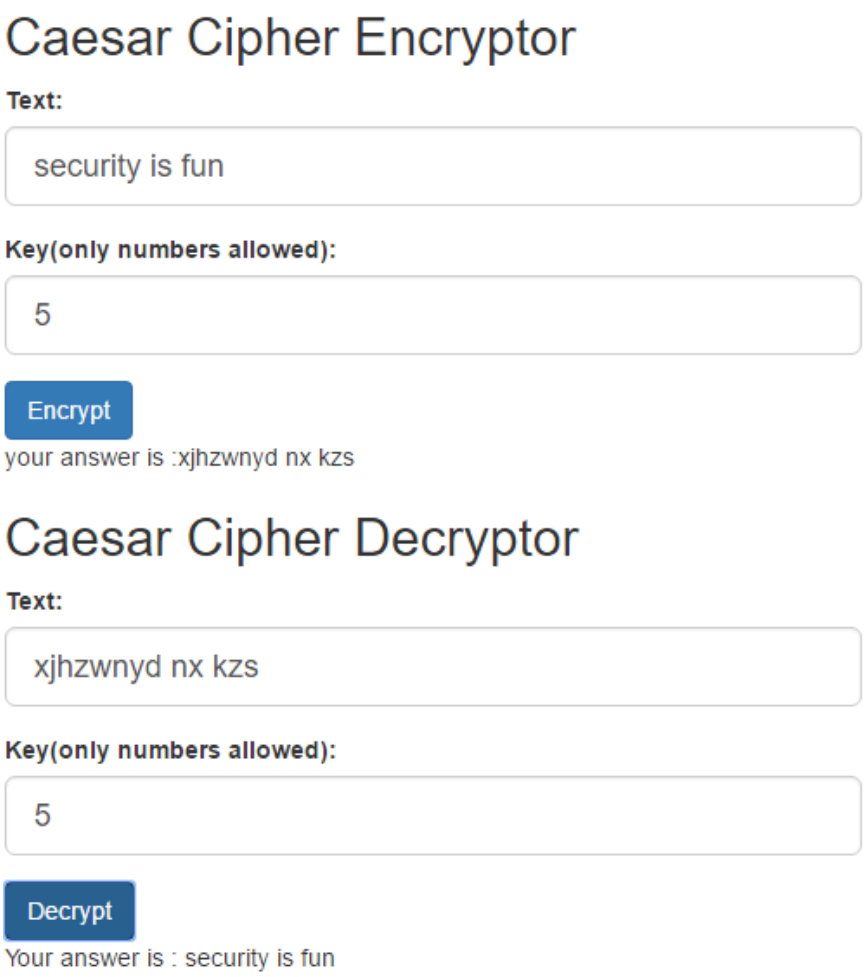
```
if ((code >= 65 + k && code <= 90) || (code >= 97 + k && code <= 122)) {
    code -= k;
} else if ((code >= 65 && code < 65 + k) || (code >= 97 && code < 97 + k)) {
    code += (26 - k);
}
```

So now instead of starting from the first letter I had to start at the “middle” 65+k and finish at the end. And when that happened I needed to subtract my shift key instead of adding it. The else part is similar in logic since we go from the beginning of the alphabet and we stop at the “middle”. And if that happens it means that the certain letter has to traverse back the alphabet.

Examples :

Security is fun , 5 -> xjhzwnyd nx kzs

i hope the code works this time since i struggled a lot ,12354 -> m lsti xli gshi asvow xlmw xmqi wmrgi m wxvykkpih e psx



Second Exercise

Encrypting

This one exercise felt really hard for me at least. I really struggled getting a good answer and the interned did not provide any help but rather more confusing stuff.

I built most of it on the framework that I already had. Especially for the html part which was pretty much the same.

The hard part occurred when I had to stop thinking about a key and thinking about turning letters into numbers. Ultimately I figured out I could just turn letters into shifting keys, depending on each letter, assigning a specific letter value that was needed for shifting.

```
function getKey(key) {
    k = []
    for (var i = 0; i < key.length; i++) {
        var c = key[i].charCodeAt(key[i]);
        k.push((c - 65) % 26);
    }
    console.log(k);
    return k;
}
```

I made a new function that would do exactly that, changing the previous k intro an array. Using a for loop iterating over each letter in the key word and getting the ASCII code value for each one. Then, in order to get the values that needs to be shifted I subtract from the character 65 (the value of A in ASCII table). This would result in a value from 0 to 25 which is exactly the number needed to shift for that specific letter. We then push that value into the array and we will use it in the encryption process.

*I use % 26 since after I tried adding “[]{}()-+\,./<>?” The algorithm was messed up .

** The reason I only use (c-65) is that in the encryption function I have uses the k variable that turns any letter into uppercase before executing the getkey function.

var k = getKey(key.toUpperCase());

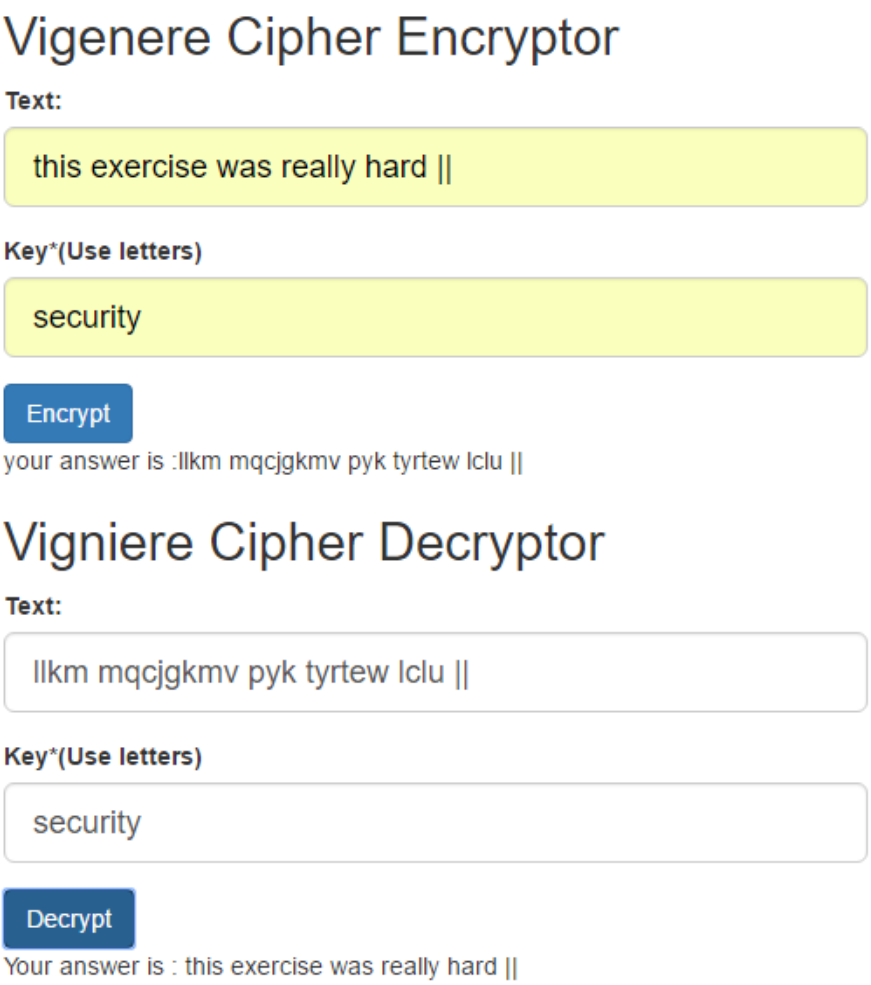
Starting the encrypt function I do what I did in the last exercises , but now I declare a length variable which is the length of my array k.

I start my for loop iterating over each letter and based on the previous example I do almost the same thing only now instead of having a simple k , since k is an array now and I need to go through each element and get the number of shifts I use ***k[i%length] instead of K*** . This iterates throughout the entire k array, and gets the values that I need to shift each time. Apart some details and adding a function this is the only big change I have from the first exercise, a change that took me a lot of time to notice.

Decrypting

Decrypting just goes in the same note as the exercise before. Everything that contained k is now changed to k[I% length].

Examples



Third Exercise

When having the code everything seems easier. The only thing left for us to do was connecting all the dots using html and javascript.

Before that I actually tried another code for checking the gcd, which seemed easier to me.

```
function gcd(a,b){
    while ( a !=b){
        if ( a > b ){
            a -= b;

        }
        else {
            b -=a;
            console.log(a);
        }
    }
    return a;
```

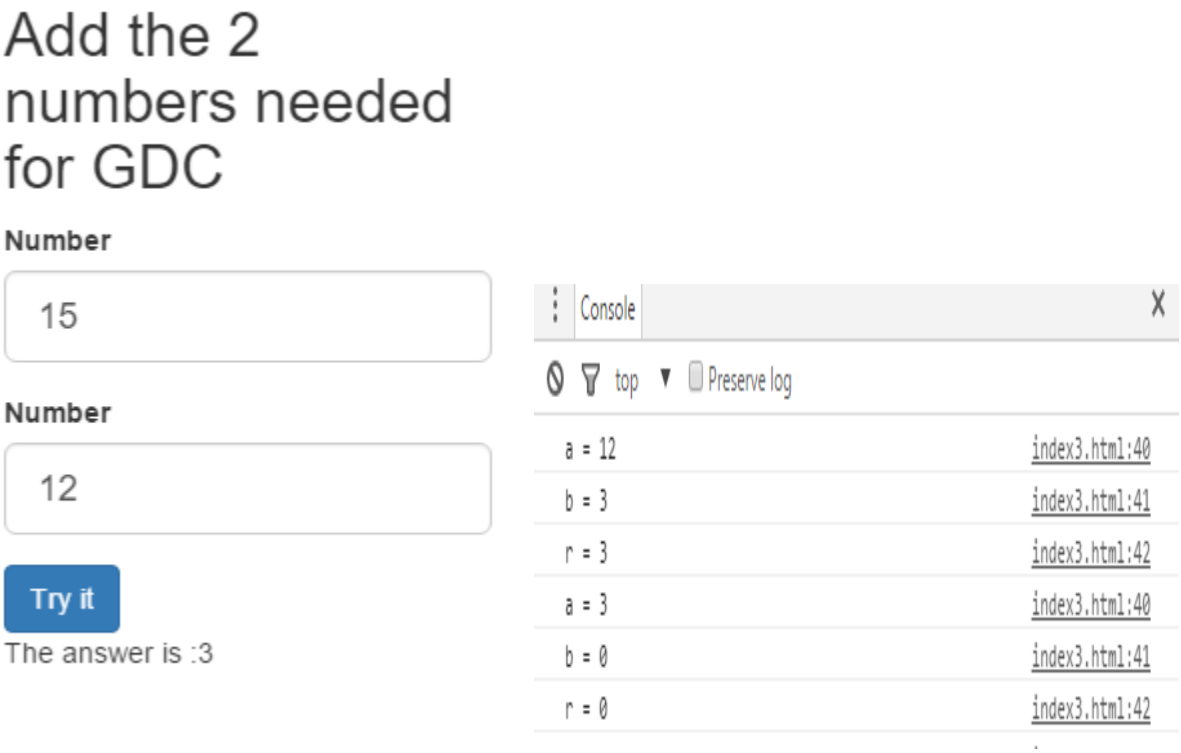
For some very strange reason this does not work inside of html elements, the function works perfectly fine outside, I checked with several numbers, but when added to my function and when having to take up the values from my input fields it just doesn't seem to work. I tried debugging but the function seems to never make it past the if statement.

Anyways , after struggling with one long enough I took the function that you gave us as an example and I tried understanding it. As easy as it was to write it, as hard it was to try to explain it on paper.

Basically we have a variable r, which is the result of modulo between the 2 numbers. The function calculates and If b is bigger than a , the 2 of them change values between each other , since

If a < b => r = a ; a=b; r=b . From these we conclude that the 2 change values between each other.

Once a is bigger than b , we use modulo to calculate the smallest common divisor that the 2 numbers have which will be assigned to the r value , and later to the b value. After a gets the value of b , the function resets and calculates again , this time using the last value of b which we know is already divisible with our r , and since it is a fix division the result will be a 0 and the function stops. This feels so much like math in 11th grade and I think would be easier to explain on paper.



In this example we have the number 15 and 12 , we use a console log to see each value after the function plays out .

So we have r = 15 % 12 which makes r = 3 , a =12 and b = 3 . The Second time we run this around we have r = 12 % 3 which makes r =0 ,so b =0 meaning the function will stop. We get our result :a =3.

Resources used:

Links that I have looked at and tried out things.

http://www.w3schools.com/jsref/jsref_charCodeAt.asp

http://www.ascii.cl/htmlcodes.htm

http://www.w3schools.com/jsref/jsref_fromCharCode.asp

http://stackoverflow.com/questions/17445231/js-how-to-find-the-greatest-common-divisor