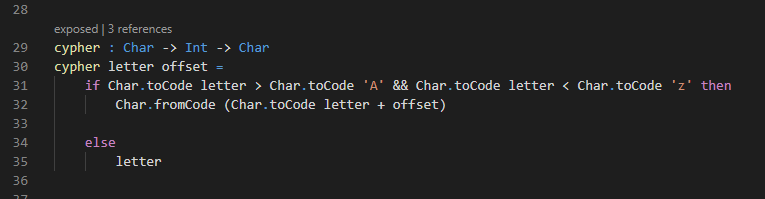
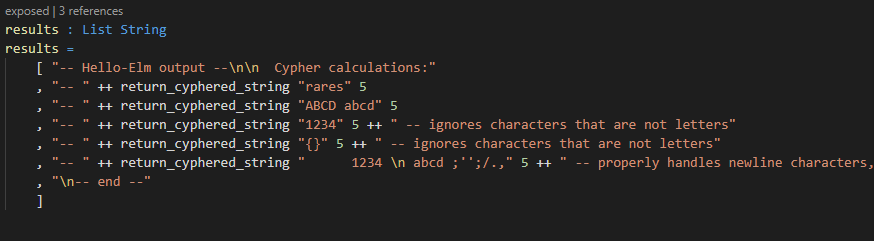
AP-Functional Programming

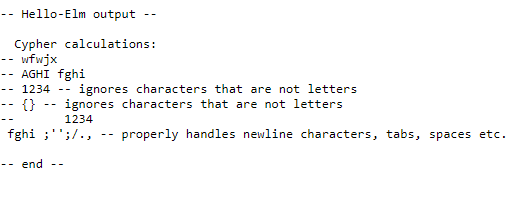
# Week1- Caesar

Assignment: Code Caesar’s Cypher for a string of characters.

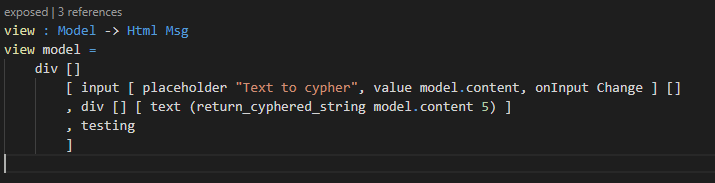


Main code that does the cypher. Takes the character as input and an offset amount and outputs the corresponding letter.

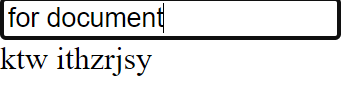




Test cases with output: Testing that only letters are getting cyphered. Also testing against newline and other special characters.

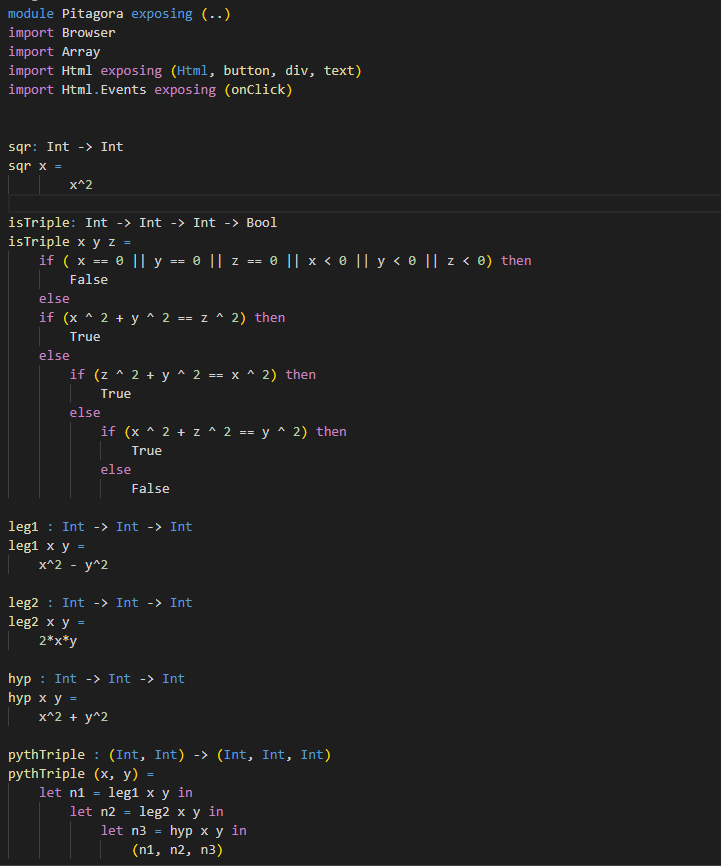


Also implemented a module that runs the cypher in real time on text using an html input in the browser.

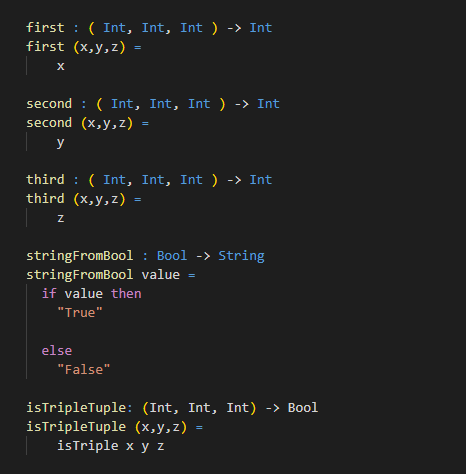


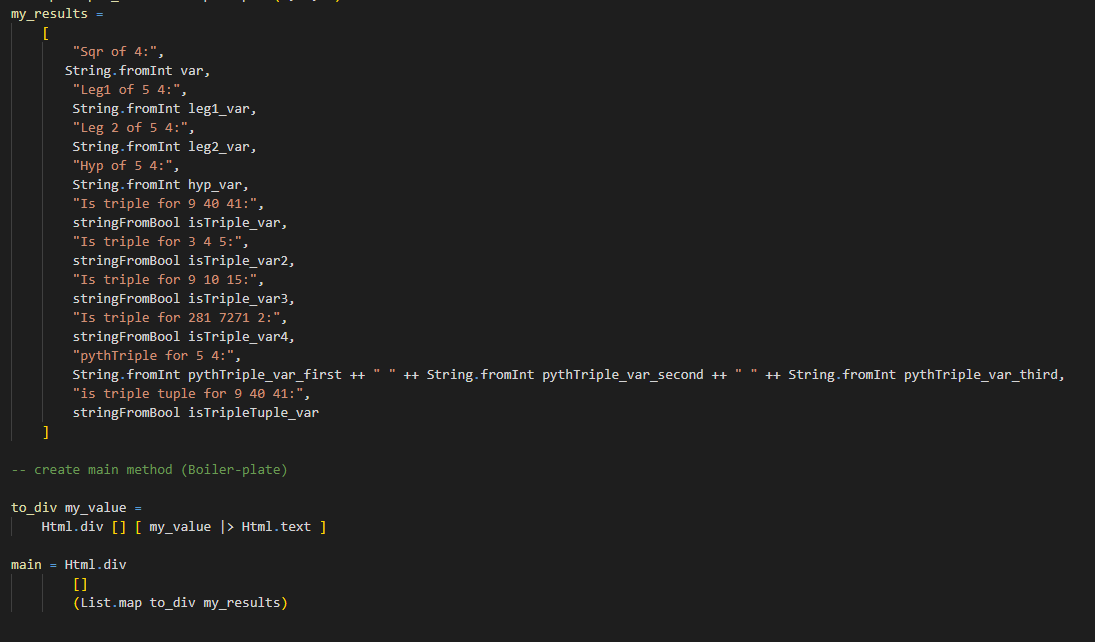
# Week1- Pythagoras Problem

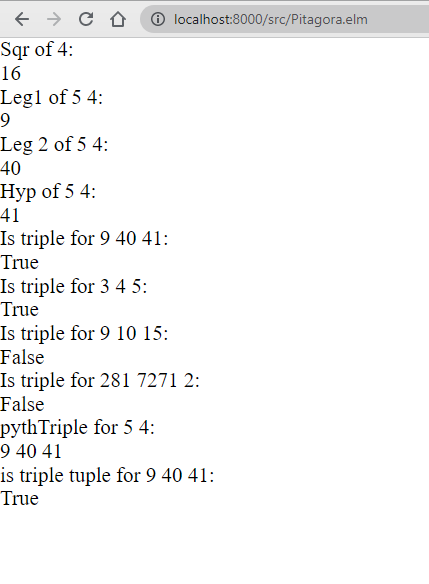
For this assignment my logic was pretty simple. Initially for checking the Pythagoras formula I thought about ordering the list and just checking if the sum of the squares of the first two numbers is equal to the square of the third, it would mean the numbers are in fact Pythagorean. As I found some issues with the elm syntax I ended up having three if statements.



Also for showing the output of the Boolean functions I had to write another function that would return a string based on the Boolean sent. That way I could have shown that my algorithm was working correctly. Also, I created functions for returning the first/second or third element of a tuple. Overall everything is working as intended and tested.



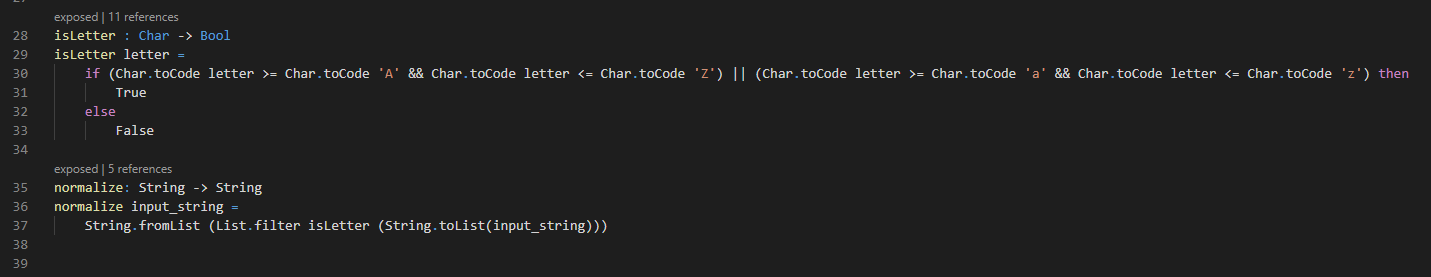




# Week 2: Caesar

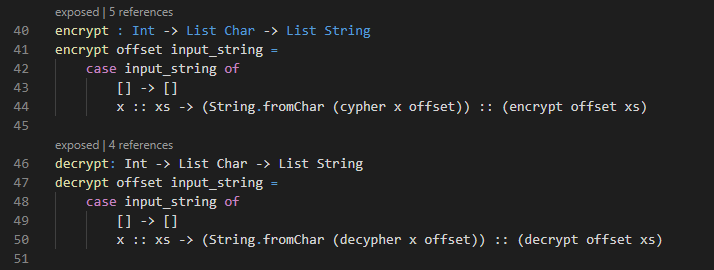
For this week, the goals were to implement a normalize() function that removes all unwanted characters from the input string, recreate the cypher without using the .map function, and make sure that the letters cannot over/underflow (when applying the offset on the ASCII code, it could be the case that “Char.toCode letter + offset” would go above/below the character set.

1. Normalize function



Pretty self explanatory implementation, I am just looking if characters in the input string are withing the character set defined (a-Z). For this function, I have used the list functions in order to go through the input string character by character, WITHOUT using map, as can be seen in the code snippet above.

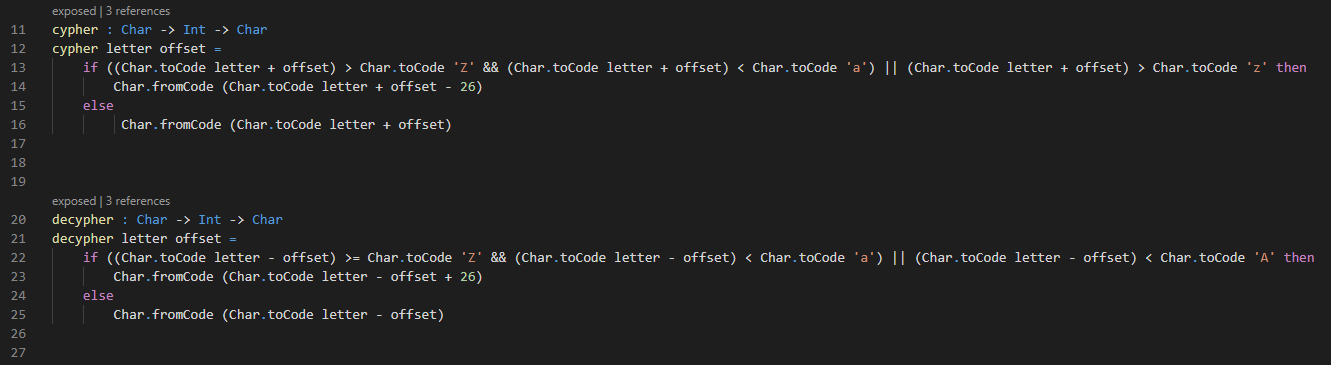
1. Encrypt/Decrypt without map



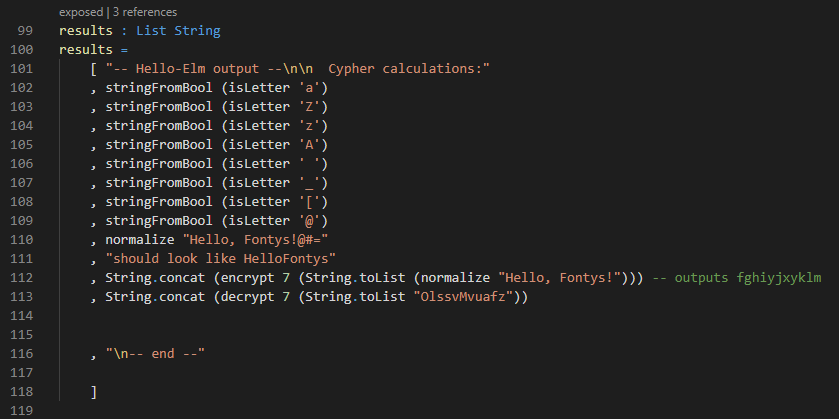
For this, I have decided to use the case distinction in the slide to implement the recursive loops that go char by char through the string. It was a bit hard to understand the implementation of recursion in elm but the slides and internet code snippets helped a lot.

1. Over/under flow in char set.

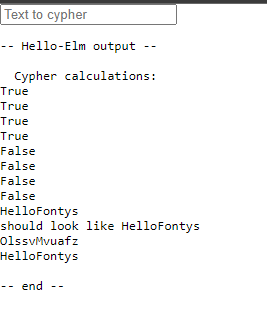
The problem was as described in the introductory paragraph that when applying the offset, the letters could go above/ bellow in the ascii table codes.



That being said, the fix was simple: check if the initial output is withing the charset – if not apply a 26 adjustment up or down to simulate looping over the charset. The number 26 represents the numbers of letters each set has: 26 lowercase and 26 uppercase, as can be seen in the ASCII table.



*Test Cases*



*Output*

Above can be seen the test cases I have written for some edge cases, and the output. I am testing for the boundaries of the isLetter function, see if the normalize function actually handles the string well and also test if the deciphered text brings me back to the input.

# Week 2: Pythagoras Problem

For this week, we had to implement two functions, one that gets a list of tuples and returns a list of tuples of Pythagorean numbers and another one that gets a list of tuples with numbers and remove the tuples that are not Pythagorean numbers. For those two functions we needed to implement once with List.map and List.filter and once with recursion.

O imagine care conține text

Descriere generată automat

List.map and List.filter implementation

For this implementation I made use of the functions created last week.

O imagine care conține text

Descriere generată automat

For the recursion implementation I still made use of some of the old functions created last week.

O imagine care conține text

Descriere generată automat

For the first function my idea was to take one tuple, generate the Pythagorean numbers and append the rest of the list calling the recursive function on it.

For the second function I had to check if the tuple is a Pythagorean tuple and based on that I will append to it the rest of the list calling the recursive function on it. Else If the tuple is not correct just call the function on the rest of the list.

Test cases:

O imagine care conține text

Descriere generată automat