Assignment 2

Klement/cgx702

1 Reflections on group work and sources

1.a

My group members are Lucas Marshall (dqr746), Noah Skovborg (tzq775) and Oliver Nygaard (lqn482).

1.b

We meet up Tuesday and talked about the assignment and the concepts we though we could use for the assignment, we focused particularly on Pascals triangle, as we deemed this the hardest part. This talk was heavily focused on getting a common understanding of the assignment, not so much actual common code. Afterwards, because of wishes from group members, most of the communication was handled through messenger, which led to the group feeling less connected, than it might have if we had meet up together throughout the week. This was this sufficient to solve the assignment, as we had all understood the assignment, before we went of and sat with it alone.

1.c

I used Fsharp.Core and slides, practices and worksheet from week 4, to look at how and which higher order functions i could use to create the array i use in 2.B.

I used work i had done in the deliberate practice and the worksheet from week 7, as a jumping of point for the higher order functions i defined for MyList in Question 3.

2 Pascal's triangle

2.a

I need to create a function which using recursion given a n and k, which are whole numbers bigger than or equal to 0, as an input pair, and calculates how many unique combinations can be made when choosing k of n. I will call this function pascal. The input for this function is a pair of integers (n, k) and the output is option type IntOrError, which is either an integer or a string, which then specifies the cause of the error.

This function used pattern matching with (n, k). The first pattern checks if these are even valid inputs. This is because (n, k) are only valid input if they are inside the triangle, meaning no negative integers are not in the triangle. We can also choose n of k, if k is larger than n, this input is also out of the triangle. The first pattern checks if any of these are true and if they are then we return Error Invalid Input. We also return Error Invalid Input, for n's bigger than 10000. We then check if we are on the edges meaning of $k = 0$ or if $n = k$, these are two separate patterns, but they both return 1. Now we get to the recursive case. Since we use the type IntOrError, for the values we can't simply get the sum using normal int operations. We instead use another pattern matching case, where we also check if the sum of the two values are negative the idea being that when we have integer overflow we simply loop around to the beginning of the 32-bit int, which is a negative number. We then check if they are both values and return the sum of those values. If either of the previous position calls return an error then we simply that error.

2.b

This function is very similar to the function from a, but now we need to fill out pascals triangle, which is a int array array, and returns the value of the (n, k) position. This function takes a pair of integers (n, k) as input and the output is an again the same option type called IntOrError, which is either an integer or a string, which then specifies the type of error. I choose to call this function pascalNoRec. We fill out the triangles position using normal integers it is only when we actually return that we return the type IntOrError depending on what the Value of the position is.

This function starts with checking if these are valid inputs, $n \geq k, n \geq 0, k \geq 0$ and returns Error "Invalid Input" if they aren't valid inputs. Next i generate a triangle using:

```
1       let mutable triangle = Array.init (n+1) (fun index -> Array.
            zeroCreate (index+1))
```

The Array.init creates an array of elements (n+1) and applies a function to each of these rows, here we create another array, filled with zeroes which has the current index + 1 elements. We need to write n + 1 and index + 1, because pascal's triangle begins at zero for both row and column. This generates a triangle which is filled with zeroes with n number of rows. The rest of this function then fills out the elements in the triangle according to the two properties. We fill a position with the value 0, if the calculated value is bigger than the max 32 bit.
I do this using two for loops. The first for loop keeps track of the rows, from 0 to n and the other keeps track of the columns from 0 to the row we are in. Inside the columns for loop, is where all the action happens. Here we check if we are on either edge of the triangle meaning, columns = rows or columns = 0, if we are on an edge we assign our current position the value 1. If we aren't on the edge of the triangle, we set our current position to the sum of the values at the position (row-1, column-1) and (row-1, column). If one of those positions has Value 0, meaning it is bigger than 32-bit max int, we set the current position to Value 0. This doesn't cause problems with the triangle being filled with zeroes to begin with since we fill out all the spaces from the top down. We fill the triangle from the top down with the value of the binomial coefficients, until we reach the end of the n'th row. This means we might have filled out some element which we didn't need to calculate the given (n, k).
After we have reach the n'te row we then return the value of the (n, k) position. If this value is equal to Value 0, then we return Error overflow.

2.c

The functions pascal and pascalNoRec, now return a option type called IntOrError, which is either an integer or a string. We now return Error Invalid Input, when we are given an input which is either $n \leq k$ or $n \leq 0$ or $k \leq 0$. This very clearly tells the person who might use this function that the input was "outside the triangle" and therefore invalid.

2.d

I have designed 7 functions the 4 that are specified in the question description and 3 additional functions. 6 of these functions all have the same type signatures:

```
1       int * int -> bool
```

These functions test if the functions have Properties 1 or 2 and that they return 0 for an input outside of the triangle. These return false if the input isn't testing for that specific test cases specification or if the function returns the wrong thing. Lets take one of these function and look closer at since they all pretty much look exactly the same.
The function hasPropertyOne function takes a pair as inputs and checks if $k = 0$ or $k = n$ if they are we run the function and return the true if the output is equal 1 and false if not. The function also returns false if $k \neq 0$ or $k \neq n$.
The only thing that changes from the function hasPropertyOne, hasPropertyTwo and the function inputOutOfTriangle is the conditions we check. For hasPropertyTwo we check if $n > k$ or $k > 0$ and if the $(n, k) = (n - 1, k - 1) + (n - 1, k)$. And for inputOutOfTriangle we check if $n < 0$ or $k < 0$ or

$k > n$ and if the output is equal 0.
I also have a function test, which simply is for my own pleasure and returns a string "PASSED" or "FAILED", depending on the result from the testing functions, which we then print in the terminal along with the pair we tested for.
We run through a list of pairs for each function that we test and print the result of. The list of inputs is unique and test the specific Property we are testing.

I have choosen values to test each Property:

```
1       let propertyOneTestValues = [(0,0);(6,0);(6,6);(10000,10000)
            ;(10000,0)]
2       let propertyTwoTestValues = [(5,2);(34,17);(10000,9999);]
3       let invalidTestValues = [(-1,0);(3,-1);(3,5);(100,-1000000)
            ;(-10000,-10000);(10000,1000000)]
```

My inputs tests each of the different cases for each of the different properties we are testing. The purpose of these tests are to make sure that we match the specifications in all cases, the test data has specifically been chosen, to see if we can break that specifications. The inputs are partitioned, so that we are only checking for one thing at a time, we don't have a function that overlaps and checks for multiple cases. We also check again with incredibly large both negative and positive values. These values go up until we reach the limit which i set, in the specification. These are to check if the specification holds up for larger values.

2.e

In the test suite we test property two with the input pair (34,17). When running the functions this returns Error Overflow because the value of the binomial coefficient is larger than the 32bit integer maximum. This means that it returns FAILED for the specific test, where we check if it is equal to sum of the function called again with pascal (n-1, k-1) and pascal (n-1,k). This means i added a pattern matching case in the hasPropertyTwo function which checks if the sum loops around and therefore is bigger than 32bit int max.
Because of the way that we partioned the input values we test for every case that the code could present us with. We test for small values, values outside of the triangle and everything else. I am 100% that these functions work for every integers below 10000 and 100000 respectively.

3 Implementing helper functions for lists

3.a

The specifications are in myList.fsi

3.b

I needed to define a function, called take, that takes and integer n and a MyList and returns a MyList of the first n elements in the original MyList. I use pattern matching on both the inputList and n. First i check whether n is less than or equal to zero and return an Empty list if it is. I then check whether the MyListist is empty and return an empty MyList if it is. If either of those base cases aren't true, then we concatenate the function call with n-1 and tail as parameters onto the head.

I need to define a function, called drop, that takes and integer n and a MyList and returns a MyList without the first n elements of the original MyList. I use pattern matching again on both the inputList and n. First i check whether n is less than or equal to zero and return an Empty list if its is. I then also check whether the MyList is empty and return an empty MyList if it is. If either of these base cases aren't true, then we call the function again with n-1 and tail as the parameters.

I need to define a function, called length, that takes a MyList and returns the number of elements in the list. I use pattern matching on the inputList, First i check whether the MyList is empty and if it

is i return 0. Then if the list is not empty i plus 1 to the return value of the function call again with only the tail.

I need to define a function, called map, that takes a MyList and a function and returns a MyList with the function applied to each element. Again i use pattern matching on the inputList. I check if it is empty and if it is i return an empty MyList. If the inputList is not empty i apply a function onto the head of the MyList and concatenate the return value of the map function called again, with the same function and the tail as paramters.

3.c

Examples are in MyList.Examples.fsx

3.d

the map function doesn't work if it is given a function with the type signature appliedFunc : string -> string and it is a int MyList. The input of appliedFunc needs to match with the type of elements in MyList. This is also specified in the specification.