Lab 2

1. Write a function cubes that will display the cubes of a list of positive integers using the Lambda expression

```
(fun n -> n * n * n)
```

Test the function for the list [2; 4; 6]

```
let main argv =
    let cubes list =
        List.map (fun n -> n * n * n) list
    printfn "%A" (cubes [2; 4; 6])
    0 // return an integer exit code
```

```
[8; 64; 216]
```

2. In mathematics, the binomial coefficient C(n, k) is the number of ways of picking ‘k’ unordered outcomes from ‘n’ possibilities. It is given by the formula:

```
C (n,k) = n! / (k!(n-k)!)
```

Implement the function C in F# and run the following test. Note that the factorials for the given values could overflow the stack so you must use the integer type BigInt of unlimited precision. This means you will have to cast integers to bigint in the computation. You can perform computations with integers too big for the 64-bit integer type by using the bigint type. bigint is not considered a basic type; it is an abbreviation for System.Numerics.BigInteger. You may want to check this: https://blogs.endjin.com/

```
C 20 5
Result 15504
```

```
let main argv =
    let rec fact x =
        match x with
        | 0 -> bigint 1
        | _ -> bigint x * fact (x - 1)

    let C n k =
        (fact n) / ((fact k) * (fact (n - k)))
    printfn "%A" (C 20 5)
    0 // return an integer exit code
```

```
15504
```

3. Vector add

vecadd adds two integer lists, element by element. Assume the two int lists contain the same number of elements

```
vecadd [1;2;3] [4;5;6]
Result list = [5;7;9]

vecadd [1; 2; -3; 4] [4; -5; 6; 7]
Result list = [5; -3; 3; 11]
```

```
let main argv =
    let vecadd list1 list2 =
        List.map2 (+) list1 list2
        // List.map2 (fun x y -> x + y) list1 list2
        // Originally the function on the previous line was (fun x y -> x + y)
        // The F# linter in VS Code suggested I not reimplement a function where no arguments are mutable
        // That led to some research and the simple function used above

    printfn "%A" (vecadd [1; 2; 3] [4; 5; 6])
    printfn "%A" (vecadd [1; 2; -3; 4] [4; -5; 6; 7])
    0 // return an integer exit code
```

```
 [5; 7; 9]
[5; -3; 3; 11]
```

4. Use vecadd to implement matrix addition. The function matadd will add two 2 x 3 matrices. Assume the matrices to be added are:

```
 M1 = 1 2 3
      4 5 6

M2 = 7 8 9
      1 2 3
```

Organized as lists of lists, the matrices to be added are

```
 M1 = [[1; 4]; [2; 5]; [3; 6]]
M2 = [[7; 1]; [8; 2]; [9; 3]]
```

The sublists represent columns of the matrices.

```
let main argv =
    let rec vecadd list1 list2 =
        List.map2 (+) list1 list2

    let matadd vec1 vec2 =
        List.map2 (vecadd) vec1 vec2

    let M1 = [[1; 4]; [2; 5]; [3; 6]]
    let M2 = [[7; 1]; [8; 2]; [9; 3]]

    printfn "%A" (matadd M1 M2)
    0 // return an integer exit code
```

```
[[8; 5]; [10; 7]; [12; 9]]
```