Lab 1

1. Implement an add function. Run the following tests.

    - add 25 15
    - add 18 34 23 19

    ```
    open System

    [<EntryPoint>]
    let main argv =
        let add in0 in1 =
            in0 + in1

        let rec sum arr =
            match arr with
            | [] -> 0
            | x :: xs -> add x (sum xs)

        printfn "%d" (add 25 15)
        printfn "%d" (sum [18; 34; 13; 19])

        0 // return an integer exit code
    ```

    ```
    40
    84
    ```

2. Write a program that uses loop. Compute the average of a list of integers. Use List comprehension to generate the List. List comprehension refers to special syntactic constructs (ex: ranges and generators) used for generating lists.

    ```
    open System

    [<EntryPoint>]
    let main argv =
        let add in0 in1 =
            in0 + in1

        let rec sum arr =
            match arr with
            | [] -> 0
            | x :: xs -> add x (sum xs)

        let rec length arr =
            match arr with
            | [] -> 0
            | x :: xs -> 1 + length xs

        let avg arr =
            match arr with
            | [] -> 0.0
            | x -> float (sum arr) / float (length arr)
        printfn "%f" (avg [1..4])

        0 // return an integer exit code
    ```

    ```
    2.500000
    ```

3. Write a function round that takes an integer argument and implements the following expression: if x >= 100 return 100 elif x < 0 return 0 else x Write a program that implements the expression using

    a. if-then-else, and then using \

```
open System

[<EntryPoint>]
let main argv =
    let round input =
        if input >= 100
        then 100
        elif input < 0
            then 0
            else input

    printfn "%d" (round 110)
    printfn "%d" (round 50)
    printfn "%d" (round -10)

    0 // return an integer exit code
```

```
100
50
0
```

b. Pattern matching

```
open System

[<EntryPoint>]
let main argv =
    let round input =
        match input with
        | x when x >= 100 -> 100
        | x when x < 0 -> 0
        | _ -> input

    printfn "%d" (round 110)
    printfn "%d" (round 50)
    printfn "%d" (round -10)

    0 // return an integer exit code
```

```
100
50
0
```

Test the program for different values of x to test the various conditions

4. Implement the function factorial using recursion. Test it for the value 5.

```
open System

[<EntryPoint>]
let main argv =
    let rec fact x =
        match x with
        | 0 -> 1
        | _ -> x * fact (x - 1)

    printfn "%d" (fact 5)
    0 // return an integer exit code
```

```
120
```