1. Asynchronous Workflows (10 points): Create an asynchronous workflow to fetch the web pages of a given URL list of size n. After the pages are fetched, their length should be printed to the console. You will need to use the System.Net and Microsoft.FSharp.Control.WebExtensions libraries.

   Test input:

```
let urls = [ "http://www.google.com"
             "http://www.udayton.edu"
             "http://www.youtube.com" ]
urls
    |> Seq.map fetchAsync // your implementation
    |> Async.Parallel
    |> Async.RunSynchronously
    |> ignore
    printfn "All urls fetched"
    Console.ReadLine() |> ignore
```

   Test output:

```
Read 47328 characters for http://www.google.com
Read 115757 characters for http://www.udayton.edu
Read 331633 characters for http://www.youtube.com
All urls fetched
```

2. Sequences/Types (10 points): Define a generic binary tree type and use yield/yield! to create a single sequence representing the inorder traversal of the tree. Then print the first and last element of the traversal. The tree type should be a discriminated union and the inorder traversal should only be around 10 lines of code.

   Test input/output:

```
let tree1 = Tree(6, Tree(2, Leaf(1), Leaf(3)), Leaf(9))
let seq1 = inorder tree1
let first = Seq.head seq1
let last = Seq.last seq1
printfn "First element: %A" first                    // 1
printfn "Last element: %A" last                      // 9

let tree2 = Tree(10, Tree(4, Leaf(2), Leaf(5)),
    Tree(15, Tree(13, Leaf(12), Leaf(14)), Leaf(16)))
let trav2 = inorder tree2
let first2 = Seq.head trav2
let last2 = Seq.last trav2
printfn "First element: %A" first2                   // 2
printfn "Last element: %A" last2                     // 16
```

3. Extend the following code for an actor to keep a running sum based on the messages it receives. Assuming the sum starts at 0, if the first message is 3, then the new sum will be 3. If the next message is 5, then the new sum will be 8, etc. Print these sums to the console. Note: It is okay for the print statements to be a bit garbled.

```
open System
open System.Threading
type MessageBasedCounter () =
    static let updateState (count,sum) msg =
        // your implementation
        let rand = new Random()
        let ms = rand.Next(1,10) // this emulates a short delay
        Thread.Sleep ms
        // your implementation

    static let agent = MailboxProcessor.Start(fun inbox ->
        // your implementation
        )

    static member Add x =
     // your implementation

let makeTask funct id = async {
    printfn "Task %i created" id
    funct id
    }
```

Test input:

```
[1..5]
    |> List.map (fun x -> makeTask MessageBasedCounter.Add x)
    |> Async.Parallel
    |> Async.RunSynchronously
    |> ignore
```

Test output:

```
Task 1 created
Task 3 created
Task 2 created
Task 4 created
Task 5 created
Number of additions: 1. Sum is: 4
Number of additions: 2. Sum is: 6
Number of additions: 3. Sum is: 9
Number of additions: 4. Sum is: 10
Number of additions: 5. Sum is: 15
```