Lab 1

Due Sunday, September 17 by 10:00pm

Handout 4
CSCI 334: Fall 2023

Turn-In Instructions

For each question in this assignment, create a project directory. For example, the source directory for question 1 should be in a folder called "`q1`". You should be able to `cd` into this directory and then run the program by typing the command "`dotnet run`", with additional arguments depending on the question.

Each program should be split into two pieces: a "`Program.fs`" file that contains the `main` method and associated program-startup helpers (if needed), and another "`Library.fs`" file that contains the function(s) of interest in the question. All library code should be in a module named "`CS334`". Be sure to provide usage output (defined in `main`) for all programs that require arguments. For full credit, your program should both build and run correctly.

Turn in your work using the `git` repository assigned to you. The name of the repository will have the form `hhttps://aslan.barowy.net/cs334-f23/cs334-lab01-<USERNAME>.git`. For example, if your username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-f23/cs334-lab01-abc1.git`.

Honor Code

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. **Be sure to tell me who your partner is** by committing a `collaborators.txt` file to your repository (5 points).

This assignment is due on Sunday, September 17 by 10:00pm.

Sanity Check: Students sometimes submit incomplete assignments, accidentally forgetting to run `git add` for all of their files. Fortunately, there is an easy way to make sure that this does not happen to you. Before you are done, `git clone` your repository to a new folder and then try building/running everything. It only takes a couple minutes and can spare you from headaches later on.

Reading

1. **(Required)** "A Brief Introduction to F#"
2. **(Required)** "A Slightly Longer Introduction to F#"
3. **(As needed)** Microsoft's Official F# Documentation

Problems

Q1. (25 points) ..... F# Types

Explain the F# type for each of the following function declarations:

(a) `let a x y = x + y / 2.0`

(b) `let b(x,y) = x + y / 2.0`

(c) `let c f = fun y -> f y`

(d) `let d f x = f (f x)`

(e) `let e x y b = if b y then x else y`

Since you can (and should) simply type these expressions into the `fsharpi` interpreter to determine the type, be sure to write a brief explanation to demonstrate that you understand why the function has the type you give. A good explanation says something like "`a` is a function that takes as input ... and it returns as output ...", being careful to explain what each of those inputs and outputs are. In particular, be sure to explain why each input and output has the type inferred by `fsharpi`.

The project directory for this question should be called "`q1`". Put your answers in a `.txt` file in that directory named "`q1`.txt".

Q2. (20 points) ..... Conditional expressions

In F#, a conditional expression looks like the following:

```
if x = 0 then
  printfn "It's zero!"
else
  printfn "It's not zero!"
```

Equality tests are written using `=`. F# does not need to use `==` to distinguish between equality and assignment because its syntax ensures that the meaning always clear.

Like everything else in a functional language, conditionals are expressions, which means that they return values. This becomes clearer when we write them inline.

```
let message = if x = 0 then "It's zero!" else "It's not zero!"
printfn "%s" message
```

Write a program that prints either `heads` or `tails` by sampling a random integer between 0 and 1 inclusive. The following creates a random number generator and calls its `Next(n: int)` method, which samples a random `int` between `0` inclusive and `n` exclusive.

```
let r = System.Random()
let num = r.Next 2
```

You should be able to run your program on the command line like so.

```
$ dotnet run
tails
```

Remember that F# always expects your `main` function to return an `int`.

The project directory for this question should be called "`q2`".

Q3. (25 points) ..... Working with arrays

In F#, arrays can be created in a number of ways. Here, we show the two simplest ways. First, one can create an array literal. For example, here we create an array literal with five elements in the `fsharpi` REPL:

```
> let arr = [| 1; 2; 3; 4; 5 |];;
```

Note that we terminate the expression above with a `;;` to let `fsharpi` know that we have completed typing our expression. When writing code outside of `fsharpi`, you do not need the `;;` terminator. `fsharpi` prints the following, to let us know how it evaluated what we wrote:

```
val arr : int [] = [|1; 2; 3; 4; 5|]
```

The second way to create an array is to use the `Array.zeroCreate` constructor. This function creates an array of length n, filled with the "default" value for the given type. For example, the "default" value for an `int` is `0`. The "default" value for a `string` is `null`. For this reason, we need to supply a type annotation, otherwise F# does not know which default value to use.

```
> let arr1: int[] = Array.zeroCreate 10;;
val arr1 : int [] = [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0|]

> let arr2: string[] = Array.zeroCreate 7;;
val arr2 : string [] = [|null; null; null; null; null; null; null|]
```

We can access an element of an array with the array index function, `[n]`. For example,

```
> let arr = [| 1; 2; 3; 4; 5 |];;
val arr : int [] = [|1; 2; 3; 4; 5|]

> arr.[3];;
val it : int = 4
```

Observe that the above expression has a period `.` before the square brackets, unlike most other languages.

For this question, write a `main` function that returns the n^{th} element in a sequence of command line parameters. In other words, write a function that starts with

```
[<EntryPoint>]
let main args =
  ...
```

that can be called on the command line like so:

```
$ dotnet run 3 fuzzy wuzzy was a bear
was
```

Here are a few tips to help you with this problem.

(a) The `args` array has type `string []`. You will need to convert the first element—`3` in the example above—into an `int`. A `string` can be converted to `int` using the `int` function, like so:

```
> let i = int "4";;
val i : int = 4
```

(b) The `main` function is expected to return an `int`.

(c) You can print using the `printfn` function. For example, `printfn "%s" "hi"` prints `hi` on the console, while `printfn "%d" 4` prints `4`.

(d) It's always a good idea to try to guard against bad user input. Your program should produce output like the following when given bad input.

```
$ dotnet run 2
Usage: dotnet run <n> <arg_1> .. <arg_n>
```

The project directory for this question should be called "`q3`". You should be able to run your program on the command line by typing, for example, "`dotnet run 1 hello world`".

Q4. (25 points) ..... Sum of Squares

Define a function `sumSquares(n: int) : int` that, given a nonnegative integer n, returns the sum of the squares of the numbers from 1 to n:

```
> sumSquares 4;;
val it : int = 30

> sumSquares 5;;
val it : int = 55
```

You should define this function recursively. Recursive functions work just like ordinary functions in F# except that you must use the `rec` keyword. In other words, your function definition should start with `let rec sumSquares ...`.

For example, here is a complete, recursive definition of a function that generates the n^{th} number of the Fibonacci sequence:

```
let rec fib(n: int): int =
  if n = 0 then
    0
  else if n = 1 then
    1
  else
    fib (n - 1) + fib (n - 2)
```

Try this out in the `fsharpi` REPL. Type it in and then end the definition with `;;` to let `fsharpi` know that you are done with your definition. Then try to call it.

```
> fib 0;;
val it : int = 0

> fib 1;;
val it : int = 1

> fib 7;;
val it : int = 13
```

Use this same approach to develop your `sumSquares` function.

The project directory for this question should be called "`q4`". You should be able to run your program on the command line by typing, for example, "`dotnet run 4`", which means that you will need to define a `main` method that calls your `sumSquares` method.

Q5. ($\frac{1}{10}^{\text{th}}$ bonus point) ..... Optional: Feedback

I always appreciate hearing back about how easy or difficult an assignment is.

For $\frac{1}{10}^{\text{th}}$ of a bonus to your final grade, please fill out the following Google Form.