Lab 2

Due Wednesday, February 19 by midnight

Handout 4
CSCI 334: Spring 2025

Coding Guidelines

Each question in this assignment should go into the appropriate project directory. For example, the solution to question 1 should be in a folder called "`q1`". When a solution is a program, one should be able to `cd` into the question directory and then run your program by typing the command "`dotnet run`", with additional arguments depending on the question.

Every 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. Library code should be contained within 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.

If any of your programs take input from the user, be sure that your program validates input: when a user fails to supply input, or supplies input that does not make sense, your program should print a usage message and return with a nonzero exit code. Users should never experience a program crash in this class; exceptions should be prevented from arising or be caught whenever bad input is encountered. Think through problem corner cases carefully.

Turn-In Instructions

Turn in your work using your assigned `git` repository. The name of your repository will have the form `https://aslan.barowy.net/cs334-s25/cs334-lab02-<USERNAME>.git`. For example, if your CS username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-s25/cs334-lab02-abc1.git`.

You should have received an invite to commit to the repository via email. If you did not receive an email, please contact me right away!

Group Programming Assignment

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. Tell me who your partner is by committing a `collaborators.txt` file to your repository. **Be sure to commit this file whether you worked with a partner or not.** If you worked by yourself, `collaborators.txt` should contain something like "I worked by myself." (5 points)

This assignment is due on Wednesday, February 19 by midnight.

Reading

1. **(Required)** "F#: The Cool Stuff"
2. **(As needed)** Microsoft's Official F# Documentation

Problems

Q1. (25 points) ..... Sum of Squares

Define a function

```
sumSquares: 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 ...`.

You may find it helpful to see a complete, recursive definition of a different function as an example. Here is 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 "`q1`". The `sumSquares` function should be put in the file `Library.fs`. 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.

Q2. (15 points) ..... Macaroni and Cheese

`yum(n: int) : string` is a function that returns a *sentence*, which is a `string` composed of a sequence of *words*. n is a positive, nonzero integer supplied by the user. *For each consecutive integer* between 1 and n inclusive, `yum` either appends the empty string to its output or a *word*. A word is a string that contains the substring "`macaroni`" if n is evenly divisible by 3 or "`cheese`" if n is evenly divisible by 5. Each outputted word in the final sentence should be separated by a single space character. The entire sentence must end with the word "`party`" and a "`.`" character. This problem must be solved recursively.

Here are the first ten outputs of the program.

```
$ dotnet run 1
party.
$ dotnet run 2
party.
$ dotnet run 3
macaroni party.
$ dotnet run 4
macaroni party.
$ dotnet run 5
macaroni cheese party.
$ dotnet run 6
macaroni cheese macaroni party.
$ dotnet run 7
macaroni cheese macaroni party.
$ dotnet run 8
macaroni cheese macaroni party.
$ dotnet run 9
macaroni cheese macaroni macaroni party.
$ dotnet run 10
macaroni cheese macaroni macaroni cheese party.
```

The project directory for this question should be called "`q2`". `yum` should be put in the file `Library.fs`. You should be able to run your program on the command line by typing, for example, "`dotnet run 10`".

Q3. (35 points) ..... Zipping and Unzipping

(a) Write a function `zip(xs: 'a list)(ys: 'b list) : ('a * 'b) list` that computes the product of two lists of arbitrary length. You should use pattern matching to define this function:

```
> zip [1;3;5;7] ["a";"b";"c";"d"];;
val it : (int * string) list = [(1, "a"); (3, "b"); (5, "c"); (7, "d")]
```

When one list is longer than the other, repeatedly pair elements from the longer list with the last element of the shorter list.

```
> zip [1;3] ["a";"b";"c";"d"];;
val it : (int * string) list = [(1, "a"); (3, "b"); (3, "c"); (3, "d")]
```

In the event that one or both lists are completely empty, return the empty list. Note that in `dotnet fsi`, calling the function as below will produce an error because F# cannot determine the type of the element of an empty list.

```
> zip [1;3;5;7] [];;

  zip [1;3;5;7] [];;
  ^^^^^^^^^^^^^^^

code/stdin(14,1): error FS0030: Value restriction. The value 'it'
has been inferred to have generic type
    val it : ((int * int * int * int) * '_a) list
Either define 'it' as a simple data term, make it a function with
explicit arguments or, if you do not intend for it to be generic,
add a type annotation.
```

To make empty lists work, explicitly provide a type for the return value.

```
> let xs : (int * int) list = zip [1;3;5;7] [];;
val xs : (int * int) list = []
```

(b) Write the inverse function, `unzip(xs: ('a * 'b) list) : 'a list * 'b list`, which behaves as follows:

```
> unzip [(1,"a"); (3,"b") ;(5,"c"); (7,"de")];;
val it : int list * string list = ([1; 3; 5; 7], ["a"; "b"; "c"; "de"])
```

(c) Write `zip3(xs: 'a list)(ys: 'b list)(zs: 'c list) : ('a * 'b * 'c) list`, that zips three lists.

```
> zip3 [1;3;5;7] ["a";"b";"c";"de"] [1;2;3;4];;
val it : (int * string * int) list =
  [(1, "a", 1); (3, "b", 2); (5, "c", 3); (7, "de", 4)]
```

You must use `zip` in your definition of `zip3`.

(d) Provide a `main` function that exercises all of the above cases, plus a few more that you think of yourself.

The project directory for this question should be called "`q3`". The `zip`, `unzip`, and `zip3` functions should be put in the file `Library.fs`. You should be able to run this program using "`dotnet run`" without any additional arguments.

Q4. (20 points) ..... Currying

Show that the types `'a * 'b -> 'c` and `'a -> 'b -> 'c` are essentially the same by defining two functions, `curry` and `uncurry`. If your definitions are correct, the following two properties should hold for all functions `f: 'a * 'b -> 'c` and `g: 'a -> 'b -> 'c`.

```
uncurry (curry f) = f
curry (uncurry g) = g
```

Define your functions in the space below. Don't forget that you can return a function definition using the lambda syntax `fun x -> ...`

```
let curry (f: 'a * 'b -> 'c) : 'a -> 'b -> 'c =
```

```
let uncurry (g: 'a -> 'b -> 'c) : 'a * 'b -> 'c =
```

To check that your work is correct, we supply the entire `main` function for your program here. In it, we we define a function `let plus (x,y) = x + y`. If your `curry` and `uncurry` functions are correctly defined, `main` should print `Result: 3` to the screen:

```
open CS334

[<EntryPoint>]
let main args =
    let plus (x,y) = x + y
    let result = (uncurry (curry plus)) (1,2)
    printfn "Result: %A" result
    0
```

The project directory for this question should be called "`q4`". The `curry` and `uncurry` functions should be put in the file `Library.fs`. You should be able to run your program on the command line by typing "`dotnet run`".

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.