.CS4402 Learning journal 7

During this week of learning, I felt it was an interesting journey that I have embarked on about functional programming.

At the high level, I gained more impression of how functional programming can work. Despite I feel it's going to take longer time to learn this language and Haskell doesn’t seem as intuitive as the other imperative programming languages like Python and java.

It seems it has a more sleek and sharp expression when comes to using very precise and concise syntax to make the code easy to maintain and it's designed in such a way that it encourages the most elegant way of writing the code with minimum redundancy.

Unlike imperative languages, Haskell uses immutable data, which means that once the variable is assigned the value will not change. It seems cumbersome but it promotes clean and predictable code as it removes the dependency of monitoring the change of state. It emphasizes the pure functions and it's always predictable.

When it dives deeper, the function composition is a powerful concept that we can freely aggregate functions to form new functions. We can format higher-order functions as we pass functions as the arguments for other functions.

Lazy evaluation is a fantastic feature that is emphasized in Haskell. The evaluation is only happening when it is needed. We can use infinite value when it needs and laziness allows us to take this advantage. Recursion is one fundamental concept that Haskell relies on.

Patten matching is a great concept in Haskell. We can destructure using patterns. We can destructure a list into its head and tail using [x: xs]

The Haskell adapted strong and static typing. This type can be specified in functions and variables, which helps to catch errors at compile-time.

Haskell uses a separate library to manage the side effects like I/O and state changes. This provides an isolation between the state and the pure functions part of the program.

Haskell provides a powerful syntax sugar to generate and manipulate lists using list comprehensions. It is very powerful for working with data.

The type classes are like interfaces in other languages. They define a set of functions that types must implement. An example can be the famous type class `Eq`, which is for types t to compare equality.

Overall, I gained some exposure with Haskell, the functional programming concept is available in different languages. These functional concepts helped me write cleaner code and slimmer codes.