CMT304 - Part 2: Functional Programming

Mohammed Shaad Mehboob Matcheswala (21065793)

CARDIFF UNIVERSITY

1]

Why the functional programming should be used?

1]. Lazy evaluation:

Functional programming(FP) performs lazy evaluation in which evaluation is only performed on an expression when the result is needed. It does not compute the values that are not to be used by the function, which eventually reduces CPU utilization and helps save time. For example, if an infinite list of prime numbers is defined and on it, a function is called which requires just five values from the list then only the five elements are computed.

2]. Purity in functional programming:

A pure function does not have side effects which means that the output would be the same all the time if the input is not changed [1]. Purity could be implemented in some functional programming like Haskell, Clean, Mercury, etc by the type signature (for example type signature could be implemented in Haskell as follows:- functionName :: input_type -> output_type) through which the side effects are controlled. If the function returns the same value in every instance this leads to the minimization of the bugs in the code.

Two arguments against using it.

1]. Recursion instead of loops:

Functional programming languages do not have do-while or for loops, it only supports recursion which makes them unique and complicated [2]. Imperative programming is widely used in most places and recursion is considered to be expensive, and therefore it is avoided due to that users need to change their mindset and adapt to recursion.

2]. Terminology problem:

For a user to apply FP, they need to know mathematical concepts as well as programming which makes FP much more complicated than other paradigms. To use it more efficiently users might need to understand advanced mathematics terminologies like monads, monoids, and functors, this makes the code even more complex for the user to understand and write.

2]

Pure functional or imperative programming paradigm

Here the paradigm of my choice is imperative programming as I feel that is less complicated than functional programming, which completely depends on an individual's perspective. Compare to functional programming, one of the qualities which make imperative programming simpler is that it provides various looping techniques like for, while, do-while, and it also supports recursion (one reason loop is supported is mutability), providing a programmer with multiple choices whereas in functional programming recursion is only a way for looping. One of the challenges which I faced in the initial stages of solving the coursework example was accessing the elements inside the nested elements through recursion.

Functional programming languages are strongly based on mathematical concepts like calculus and algebra. Plenty of people do not have a better understanding of such mathematical topics including myself and hence programs written in a functional programming language are not elegant and easy to understand. According to my experience, the codes written in imperative programming are easy to write and understand.

Imperative programming is the most prevalent programming style used today as a result if a programmer could receive plenty of help from open-source platforms. On the other hand, functional programming is complicated and not everyone has expertise in it so there are fewer sources available to learn from. Also, imperative programming languages like Java have widely built libraries compare to Haskell.

To conclude, there are plenty of advantages of using FP as it uses memory efficiently despite using recursion, no side effects, lazy evaluation, debugging the code is easy, and many more. The problem mentioned in the coursework could be solved by both the paradigms but in my opinion, imperative programming would be much more appropriate as a wide range of support could be obtained from open source platforms, plenty of libraries and less complex coding as people with fewer knowledge would be able to solve or understand the code.

References:

[1]. Moya, D., 2022. *Purely functional programming - Wikipedia*. [online] En.wikipedia.org. Available at: <https://en.wikipedia.org/wiki/Purely_functional_programming#Data_structures> [Accessed 11 May 2022].

[2]. Alexander, A., 2022. *Disadvantages of Functional Programming | alvinalexander.com*. Alvinalexander.com. Available at: <https://alvinalexander.com/scala/fp-book/disadvantages-of-functional-programming/.> [Accessed 11 May 2022].