Student Name: Joe Kirkup Student Number: 29026253

Functional Programming – Evaluation

The Functional Paradigm

What is functional programming?

Functional programming (hereinafter FP) is a programming paradigm which encourages a few core concepts: division of code into 'atomic' functions, immutability of data, and prevention of side effects.

While none of these functions necessarily guarantee anything, as programmers still have freedom to write whatever code they want, they aim to, in turn:

- Division of code into 'atomic' functions (functions which only do one operation on a piece of data) means that code is easier to debug and write unit tests for, as it is not only clear what each function does, but also easy to quantify given that the data that went in should only have been modified slightly.
- Immutability of data ensures that once some data has been assigned or transformed, it will stay that way, unless we explicitly call some function and overwrite its value. This means that true functional code can be mathematically proven for all possible inputs, which is a very useful concept – when working with critical real-world systems, it is invaluable to be able to know for sure exactly what the code will do in a certain situation. Doing the same with a typical imperative program requires 'brute forcing' using dummy inputs to see what the code will do, which is not only harder to do but also less certain (foreseen circumstances could be incomplete).
- Prevention of side effects is as straightforward as it sounds – printing, writing files, writing to a database, an API call, any kind of input or output are regarded as dangerous and unwelcome in FP. This ties in strongly to the concept of proving code discussed above – if there is any kind of variable input, there is a risk of the state of the program moving into unknown territory.
  At first, to a programmer with a history of writing imperative code, this may sound insane – how will my code do anything useful if it can't interact with any data!? Well, functions can still take input, but they should only do so through their parameters, and they should only output through their return value.
  Obviously for programmers or users to interact with the code at all, a few side effects are unavoidable, but FP developers approach these carefully; an ideal but realistic functional program should be called with some initial arguments, complete all of its computation, *then* output results of the computation.

How does FP differ from imperative programming?

While there are numerous languages which are designed specifically with functional programming in mind, elements of FP are present in all modern languages, and the line between the two paradigms is often blurry, especially given that many modern languages implement imperative and functional concepts.

Clojure

A description of the language

Clojure is

Integration with Java libraries

Comparison to ____