Are Functional Programming Language Becoming Popular Again?

Pooja Makula[1], *SridharChimalakonda*[2]
Indian Institute of Information Technology, Chittoor, Sri City, A.P., India
pooja.m13@iiits.in[1],*sridhar_ch@research.iiit.ac.in*[2]

***Abstract*—The use of Functional Programming languages has rapidly increased over years. With huge amounts of data to be processed and use much more minimalistic codes Functional programming languages have become much more popular. In this paper we aim to bring out the**

***Index Terms*—**

I. Introduction

The advent of Functional Programming dates back to 1950s, with programming languages like Lisp, FP and many more. However, they were dominated by imperative languages due to the ease imperative languages like C offered and also due to the lack of competent machines that can effectively illustrate the benefits of functional programming. With multi-core computers, powerful concurrent programming constructs, cloud computing services and many exhaustive applications involving nodes and powerful servers all connected across the world, the notion of functional programming is finding its relevance and importance. From the very basic Lambda calculus to finding existence in many current languages, a huge upsurge can be observed in functional programming. In order to understand the extent to which functional programming has evolved into regular programming styles, this paper takes the approach of comparing functional languages and procedures that evolved with time.

II. Imperative Programming vs Functional Programming

Programming styles over the years have evolved a lot along with the growing complexity of the code and the computers. Imperative programming is such in which the code can directly manipulate values of variables modifying them to acquire the required state. The core difference between the two models lies in the approach the coder takes to solve the problem. While in imperative programming, the prime focus is to attack the whole problem by even hard coding how to perform every task. This makes every part of the code very specific to the problem and cannot be generalized any further. Whereas, in Functional programming, the core focus is to develop robust, generalizable programs which are easier to analyse, test for performance or bugs or for reusing the code. Some fundamental concepts which differentiate between both the coding techniques are discussed below.

Traditional Imperative style of languages are based around the idea of a variable as a changeable association between a name and values. These languages are said to be imperative because they consist of sequences of commands
Basically, each command consists of an assignment tampering a variables' value. Assignment here involves working out the value of an expression and associating the result with an identifier:
In a program, each command's expression may refer to other variables whose values may have been changed by preceding commands, enabling values to be passed across commands. Whereas, Functional languages are based on structured function calls. A functional program consists of an expression with many nested function calls:
Every function receives values from and passes new values back to the calling function. This process is known as function composition. So, in functional languages, a name can never be associated with more than one value, because its the values that are passed across functions, not their place holders.

III. Impact of the Functional Programming languages

Functional programming has enjoyed a great surge in the recent years as can be observed in the proliferation of books and conferences, in the rapid growth of languages like Scala and Clojure, and also in the range of applications being implemented in these languages as observed on git, bitbucket etc.

These days, no language designer would dare announce a new programming language without supporting "functional programming". This can be seen clearly as even the conservative, mild-mannered, Enterprise-certified Java is getting lambdas and even monads these days. Some of the reasons for the increasing acceptance of these languages can be traced as follows. Fot the past few decades as new problems and demands arose, people started developing new languages specific only to their domain or only to that particular application also at times. They chose imperative languages because they appeared more easier to develop than the strong functional languages which require a lot of planning before directly attacking the problem. While using Imperative languages like C, programmers have full freedom as they can assign values directly, even at the processor levels, and there is nothing stopping them from directly writing the code. Whereas if one has to code in a functional language, the programmer has to first understand the requirements properly,

divide the into modules, then implement each module with no dependencies involved, which requires a lot of brain work. Also, while working with imperative languages, programmers are so carried away by the ease they present that they fail to realize the risks at each point of the code. A lot of time is wasted to test the code and debug it. Finding the cause of the error and all the variables and constants that depend on it is a tedious and a costly task. Also, with the rapidly increasing sizes of codes and machine critical systems, large IT systems depending largely on the code, it is becoming very complex to identify the root cause of an error or fallacy and then correct it in such a manner that the rest of the code is not affected.

As software becomes more mature to be called a profession, programmers have to adopt ways that lands them in a safer place i.e, with crisp, error-free, easier to test and analyse type of code. Another aspect of functional programming that is making it increasingly popular is the kind of ease it offers. Dividing the code into modules, taking care of the performance of each function makes the system more robust, easier to handle and also easier to operate on large systems. With the advent of the Internet, almost all of the applications developed and used today involve more than a single computer, or at least more than a single core, which is making concurrent programming increasingly important these days. With such a strong necessity, the notion of dividing the problem into independent parts and then solving it is the way to go and functional programming makes it easier. Also, it brings ease because when minor modules are handled by the language itself, then the work of the programmer becomes easier. If minor issues like sorting is already implemented as a module with the best performance possible, the programmer can focus on more complicated stuff, the language becomes less of the problem and he starts concentrating on actual problem statement.

Another reason for this upsurge could be the ubiquity of programming these days, even children at schools are learning how to code. Software development is not about implementation, but about problem solving. Implementation can also be done by the machine with code generators, etc. So, in order to develop software which the developer can guarantee, his mind work, the algorithm and the choices, the decisions he takes during the implementation are the most critical. Functional Programming also helps in making code more understandable and organized.

IV. Applications of functional Programming Languages

Certain Characteristics of Functional Languages which is making functional languages more relevant in today's world are:

- Higher-order functions: These functions allows passing functions as input to functions, or return functions from functions, which helps in factoring out a lot of duplication.
- Immutable Data Structures: These are often used in FP, they free the programmer from having to constantly worry about what some particular code will do to the data passed to it.
- Strong types: These appear in many functional programming languages (but not all). They tell us more about statically-proven properties of the code.
  Pure functions: These are functions that don't have side effects (i.e. their output is a deterministic function of their input), are much easier to understand and test, because one need not wonder if the function's behavior will change based on any hidden state.

V. Experiments and Results

In order to analize the usage and trends of the functional programming languages, and the different paradigms of these languages, the following experiments have been done.

- Survey on the trending projects in Github: