2025

CS 4402

Comparative Programming Languages

LEARNING JOURNAL 1

SANA UR REHMAN

INSTRUCTOR: JIM CASALE

INTRODUCTION

During this week I have learnt about the foundations of programming languages, their history and how they have been proven effective in reshaping the expressions of problems in codes. I studied various types of languages such as imperative, data-oriented, object-oriented and non-imperative languages, as well as the theories of computation rooted in the von Neumann architecture, Turing Machine, and Lambda Calculus. This gave me insights into the functionality of these languages and how they have made modern computing possible. Ine the world of computing language my knowledge was greatly broadened after this week's readings.

DIFFICULTIES FACED

While these concepts were interesting and attention catching, some concepts were a bit difficult to grasp such as distinguishing between paradigms and understandings of the practical implications of these languages. As I was already familiar with imperative languages like C and Java, the paradigms of non-imperative languages such as functional and logic programming were more abstract for me, as they emphasized "what" to compute instead of "how." Another problem I faced was the implementation of the theoretical knowledge of models like the Turing Machine and Lambda Calculus to real-world programming practices. Initially they seemed highly mathematical but as I practiced more and reviewed more examples, I was able to see how they formed the basis of modern computation.

ACTIVITIES PERFORMED

I completed the readings that were assigned and reflected on differences in imperative and non-imperative languages in my discussion post. I explained that imperative languages are interested in state

changes and step-by-step instructions, while non-imperative languages are founded on mathematical functions and logical expressions (Sebesta, 2016; Watt & Brown, 2000). The process of writing this post forced me to connect the theoretical to practice, e.g., system programming in C versus data analysis in Haskell. I also did the self-quiz, in which I was forced to identify paradigms and the roots of computing. The quiz reinforced my knowledge by forcing definitions and examples to use, which helped differentiate among paradigms.

REFLECTION

What struck me this week was the extent to which programming languages are influenced by theoretical models created decades ago, when there were no computers yet. Learning about Alan Turing's idea machine and Alonzo Church's Lambda Calculus reminded me of the impact of theoretical mathematics on the languages that I currently work with. This insight prompted me to have greater appreciation for the intellectual roots of computer science.

The most challenging part was programming with non-imperative languages in a functional mindset. I'm used to coding programs with loops and assignments, so recursion and immutability were foreign at first. It showed me that stretching my thought process when solving problems will take some getting used to but will eventually make me a greater polymath programmer.

I know that I am building theoretical and practical reasoning capacities. Theoretical knowledge is built through the knowledge of paradigms and computational models, while practical reasoning is built through considering what paradigm is appropriate for specific areas. As a learner, I am realizing that I am connecting new information to what I already know, and this helps in putting abstract things into more concrete contexts.

As regards the application, I can see how studying different paradigms makes me ready for actual problems involving the universal use of hybrid approaches. For instance, functional programming practices

would render code more transparent to data transformations and yet imperative approaches remain a necessity in low-level control. This balance highlights the significance of adaptability in programming.

CONCLUSION

After this week's reading I can say my understanding of technical and theoretical perspectives of programming languages was deepened. I got more clarity of understanding the paradigms and their practical uses in several cases, and how foundational models like the Turing Machine and Lambda Calculus shape computing till today. While I found functional and logic-based paradigms more difficult to grasp, they challenged me to think differently about programming. In nutshell not only have I expanded my toolkit through learning these languages, but it has also given me a better insight into conceptualizing problems and solutions in computing.

REFERENCES

Sebesta, R. W. (2016). *Concepts of programming languages* (11th ed.). Pearson. https://www.pearson.com/se/Nordics-Higher-Education/subject-catalogue/computer-science/sebesta-concepts-of-programming-languages-11e-ge.html

Watt, D. A., & Brown, W. F. (2000). *Programming language design concepts*. Wiley. https://www.wiley.com/en-us/Programming+Language+Design+Concepts-p-9780470853207

Word count: 670