

# Preface

We are living in a very interesting time, in terms of technological advancements among other things. In the last 60 years, humanity has made tremendous progress in the field of computing. High level programming languages appeared in the late 1950s, but there were no microprocessors available. One can imagine how tedious it could be to program. With the invention of microprocessors in the late 1960s, this field got a great boost. Along with hardware developments, new high level programming languages started emerging. Among those are Pascal, Smalltalk, C, etc. Some of these languages are still widely used. One of the interesting facts about these high level languages is that they had niche application areas, even though they were widely projected as general purpose languages, including in the textbooks. For example, Pascal was leaning toward education, C was primarily developed to ease operating systems developments, Fortran was for scientific computing, etc. Whatever the domains, each of these languages has shaped the history of computing.

Today, shapes, sizes, and capacities of computing devices are much different than 60 years ago. In the last 20 years, computing has shifted from localized client-server environments to world wide client-server environments. There are several catalysts for this change, including the invention of the world wide web, the advancements in integrated circuits, and the innovations in device manufacturing. One of the catalysts, which doesn't get enough credit, is the advancements of programming languages. Developers are truly enabled by language innovations. The innovation that JVM brought, has greatly impacted software engineering in general, and Internet based applications development in particular. Being able to program, without worrying about incompatibilities with hundreds of vendor specific architectures, is a great relief for software engineers.

One of the important aspects of advancements, not discussed enough, in industrial settings is knowledge evolution. I have had the privilege of working for the world's largest (non-profit) organization, the United Nations, which gave me an opportunity to work with people from almost every major cultural background. Also I have worked for some of the largest and finest for-profit organizations in the world. In addition to this, I was fortunate enough to be part of some of the finest universities in the world, either through academic programs, or by working directly with aca-

demicians in unique settings. This experience allows me to make some inferences on knowledge evolution, specially in the field of software engineering. The first inference is that changing a programming language is not pleasant, specially for professional programmers. The second inference is that adapting a new language to the level that one can sell the skills has strong limitations. The third inference is that the popularity of a particular language is strongly affected by the industry–academic loop, directly or indirectly. To be precise, how a language is taught, where it is taught, what kind of learning materials are available, etc., determine how popular a language will become.

Dr. Martin Odersky, the creator of Scala, has done a great job of a language innovation. Scala not only provides an opportunity to program in multiple paradigms, but also makes developers more productive with today’s computing infrastructure. Most of the earlier languages were not designed to program in distributed environments. Also most of them were not designed to evolve. One of the fascinating aspects of Scala is that it can be grown, based on developers’ needs. Remember, for every genius, there is a limit to how many languages one can become expert in. It is pretty much like the case of natural languages. We can recall ourselves, how good we are in our mother tongue and how good we are in other languages. Do we struggle? The Scala creator has gifted a beautiful solution to the world of computer programmers.

With this book, I have tried to enrich the learning aspect of Scala. I strongly believe that Scala can be a great first programming language. There is no need to lean on any other programming language in order to learn it, except the environment, which is JVM for a good reason. The approach that I have taken in this book was primarily inspired by my first hand observations of how professional programmers master a new technology. Also it is partly influenced by my experience of teaching undergraduate computer science and engineering students, over half a decade. I have tried to present complete and runnable programs, whenever possible. In the last 12 years, I noticed professional programmers learning faster by tweaking existing programs. This was true for my undergraduate students as well. Tweaking allowed learners to build self-confidence, which I found was the basis for the majority of learners for their long term pursuits. This book can be used, both at an undergraduate level and at a graduate level, to teach a first programming course. Also it is a great companion for professional programmers planning to switch to Scala.

Each chapter has review questions to reinforce your learning, which makes you ready for problem solving. You will find yourself adequate and self-confident, thereby keeping you away from the path of frustration. I have seen many great professional programmers being frustrated while learning a new language. Also each chapter has problems to solve. These problems are much closer to what you will need when you embark on your career as a professional Scala programmer. The process of solving problems expands your knowledge boundary; you are moving from reinforcing to developing salable skills.



<http://www.springer.com/978-3-319-69367-5>

Programming with Scala

Language Exploration

Upadhyaya, B.

2017, XIX, 194 p. 67 illus., Softcover

ISBN: 978-3-319-69367-5