Preface

Author(s)
Silvio Peroni – silvio.peroni@unibo.it – https://orcid.org/0000-0003-0530-4305
Digital Humanities Advanced Research Centre (DHARC), Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

---

I want to start with a clear disclosure: the main aim of this book is to introduce two specific Computer Science topics, namely **Computational Thinking** and **Programming**, to students **in the Humanities** with **no prior knowledge** on them. In particular, the book presents the *essential* and *practical* aspects of these topics – abstraction, algorithms, data structures, programming languages, which are among the primary knowledge, methods, and tools, characterising Computer Science as a discipline.

The idea of writing this book came from the need of providing a free textbook to the students of the Computational Thinking and Programming course within the Second Cycle Degree in Digital Humanities and Digital Knowledge at the University of Bologna. The topics introduced in this book are usually delivered in three or four distinct courses of a standard Computer Science Bachelor degree, I organised its chapters to provide enough tools to students to enable them to start their computationally-related career within a Digital Humanities (DH) curriculum. As you can imagine, having the constraint of a 30-hour course to introduce these topics, there is no space for an in-depth presentation of all the philosophical aspects related to Computational Thinking – and other Computer Science topics in general. However, each chapter provides pointers to enable curious students to explore and increase their knowledge on these topics – something that students in the Humanities can do brilliantly.

Another essential clarification: this is *not* a book about Python. As you may notice, there is no "Python" word in its title. The book is meant to introduce how to use a *language* to *communicate with* and *instruct* an information-processing agent (being it a human or a machine), which is one of the core tasks of Computational Thinking. In particular, the book presents general notions and methods – namely *algorithms*, *data structures*, and *algorithmic techniques*. In Computer

Science degrees, they are usually introduced using pseudocode, i.e. an informal and high-level description (generally for human consumption only) of the steps of an algorithm and the methods that characterise the operations of data structures. Programming languages enter into the game when we want to instruct an electronic computer in dealing with these aspects.

After an initial presentation of flowcharts as a language to devise algorithms, the book adopts Python as a *representative language* to interact with an information-processing agent. I chose to use Python in this context due to its current use in real-world projects, particularly those related to the Humanities. Of course, several Python-related notions are briefly introduced in the book, since they are functional to the understanding of the topic presented in each chapter. However, these notions are generalizable in any programming language, despite the way Python implements them. Therefore, it is up to the students to get more in-depth knowledge about Python (if needed) by reading one of the free Python books suggested in chapter 4, such as the wonderful "How To Code in Python" by Lisa Tagliaferri.

The book introduces all the relevant topics mentioned above – algorithms, data structures, and algorithmic techniques (i.e. the methods of computing), exemplified by using a particular programming language (i.e. Python) – by iterating on basic *computational problems* (find an item in a collection, sorting a list of items, solving a solitaire, etc.). Sometimes, the book addresses the same computational problem with different techniques to show how it is possible to provide better solutions to it (e.g. more efficient strategies in terms of time/resources spent to address the problem) depending on some specific premises.

Often, students feel lost at the very beginning of their learning path on computational topics. Questions like "from where do I have to start to develop an algorithm?" and "how can I check whether an algorithm does what I want?" arise when students approach algorithms for the very first time. Thus, to support students, the book devises a development methodology for algorithms and, consequently, their formal implementation using a programming language, that recalls the well-known *test-driven development* technique in Software Engineering. While, on the one hand, such a methodology provides the answers to the questions above, on the other hand, it also introduces good practices to follow for developing software. In principle, the adoption of such a methodology and the application of the methods of computing mentioned above enables students to write a piece of work (being either an algorithm or an entire software) which is efficient, effective, robust, and maintainable in the long term.

As anticipated, this book is a textbook. Thus, all the chapters are accompanied by exercises to practice the various topics introduced. The keys of these exercises are available on the website of the book at https://comp-think.github.io and are organised by chapter. Also, the website includes additional exercises which focus on enhancing the two main activities that concern the learning of a new language: *understanding* something written in a specific language, and using such a language to *develop* new work. Remember: there can be different ways to solve an exercise and the solutions provided present just *one* possible way.

In addition to exercises, each chapter includes a brief description of a “historic hero” who is functional to discuss the main topic addressed in that chapter. These heroes are not necessarily computer scientists. Indeed, some of them are even writers. The book uses them to show that essential aspects such as computation, data structures, recursion (a fascinating concept I did not introduce in this preface that characterises our daily life), can be found in places in which they are not usually expected to be. Of course, this is not an original idea at all. The use of people outside the Computer Science domain to introduce Computer Science topics has been already adopted in the past and has shown to be very useful to introduce such topics to a varied audience.

Sometimes, it happens to hear whispers of despair like “algorithms, computations, programming, etc., are something which I cannot handle – indeed, I did not like mathematics at the high school”, “I am not able to abstract a situation by using formal tools since I am not a formal person”, and even “I cannot learn a programming language”. These are just **prejudices**. First, several of the things the book introduces are something that all of us learned and digested in our experience. It is only a matter of identifying them again to reuse them consciously. Second, we have a growth mindset. Intelligence is a muscle, it can be developed, and yes: we can learn anything if we truly want to, including Computer Science topics, despite what others say. Third, a programming language *is* a language, such as Italian and English. To properly handle a language, we need time and practice. Trying, making mistakes, correcting misuses, talking with others which are more fluent than us in that language, is critical to learn it properly. Programming languages are no exception.

Let me conclude with a final motto from a famous movie, which describes the right attitude to adopt for addressing the topics of this book: “anyone can cook, but only the fearless can be great”. Try. Make mistakes. Learn. Repeat.