Preface

John M. Morrison

July 29, 2015

Open-source technologies have greatly democratized computing and put vast volumes of knowledge and a bounteous wealth of learning opportunities in easy grasp. Todays modern computing environments make learning to program easier, more exciting and more fun than ever. Even a fairly old PC can be equipped with a distribution of LINUX and it can become a powerful tool for learning programming and for creating working applications.

There are many good LINUX distributions available; Ubuntu, Fedora and Linux Mint are all excellent and are all freely available. You are strongly encouraged to look into these.

The reservoirs of resources available on the web are vast. This book aims to be a starting point for the budding computer scientist,, or for any bright and curious person who wants to know more about computing. We assume you are curious about computing and interesting in furthering your knowledge. As you progress, you will learn about on-line resources and how to use them to your advantage.

The reason for choosing the Python language is that Python is a sophisticated language that makes programming for beginners simple. Its simple grammar and dynamic typing make it a nearly ideal language for someone wanting to embark on a study of computing. Python is a good medium to use to acquire the fundamental ideas of computer science. Mastering the ideas in this book will ready the reader for usefully reading and learning from the avalanche of resources available on the Web, as well as tackling more sophisticated treatments available in dead-tree format.

At the same time, Python is a very powerful tool which is used by organizations such as NASA, Google, and YouTube as a rapid-development tool. Python has many extensions: numpy, wxPython, scipy, ipython and pygame that allow someone who understands the core ideas laid out in this book to do sophisticated computational exploration of data, scientific phenomena, as well as a means to create graphical applications.

What this book is not This book is not an encyclopaedia of Python. It is not a reference on LINUX or on HTML5 and CSS for that matter. It is not an exhaustive technical reference of any ilk. It is unabashedly designed as a learning tool.

Absolute completeness of coverage is not the goal here. This book is meant to blaze a trail through the world of computing. We make a concerted effort to keep the conceptual spoor compact and easy to follow. It is intended to be a good tool for selfstudy. If you acquire the ideas in this book, you can then profitably read many of the references listed here to advance your study. This book is meant to open many doors for you to furthering your knowledge.

We do not promise to inject the ideas of programming into your brain without some serious effort on your part. This is not possible. You will need to write lots of code, break it, and fix it. You should work the exercises as they arise; they are designed to give you an opportunity to apply new ideas and they sometimes actually teach new ideas. Some of these exercises (gasp!) require you to take out a pencil and paper and fool with numbers. Dont skip these! You must be an active participant in the enterprise. You will need to learn how to browse various sources of documentation on the web. Making aggressive use of these will greatly enhance your experience as you work through the material.

For whom this book is written This book is for you if you are an open-minded inquiring person who finds problem solving fun and exciting. Unapologetically, we have the goal of this book being a learning tool; it was conceived and designed with that goal in mind from the very beginning. The ideas in this book have been inflicted on a wide field of students from the North Carolina School of Science and Mathematics.

No prior knowledge of computing or programming languages is assumed. However, it is a good idea to be proficient in the basics of using a modern computer before you begin. Said proficiency is very modest; it includes such things as being able to open, manage and save files. The completely uninitiated could master it in an afternoon with a little effort and a simple book on using their PC.

Approach Chapters 0, 1, and 2 constitute a unit of preliminaries that help gently introduce the ideas of computer science. Chapter 0 explains very simply the inner workings of a computer that are helpful to understanding the the significance of basic operations we all perform on a daily basis on our computers. It also introduces the idea of algorithm, and uses algorithms to study the representation of numbers.

Chapter 1 introduces the LINUX operating system. Learning about and using the commandline interface for computing is an exercise all new programmers should experience. This interface is simple and powerful, and it is very

frequently used in the professional programming world. Learning a commandline interface will yield new insight into using graphical interfaces like Mac or Windoze. Mac users who read this book will find themselves using their terminal tool more often after reading this book.

Today, obtaining access to Linux is simple. If you do this you can have a graphical desktop and create terminal sessions directly on this desktop. There is a distribution of Linux, Xubuntu, which can be obtained from the Ubuntu website, which can give new life to an older computer you may have sitting idle because it wont run the latest version of Windows. This software is entirely free; you are encouraged to explore the Ubuntu website, `http://www.ubuntu.com`. Ubuntu runs very well on a most desktops and notebooks. You can download a live CD, which is bootable, and from which you can test-drive a Linux desktop. Windoze users can also do a speical install of Ubuntu that is quick and easy.

Chapter 2 introduces the creation of well-formed HTML5 web sites. This introduction brings to the reader some of the core ideas of programming and problem-solving without all of the complexities. Browsers respond visually to the code they read; this gives the reader a way to engage in the problem-solving process visually.

Tags in HTML5 introduce the idea of delimitation; determining where things begin and end and how they begin and end relative to other objects is important. The use of the HTML5 validator in this process is an important one. Ferreting out errors raised by the validator requires developing the thinking techniques of a good debugger. This activity teaches the participant how to intelligently read and use error messages to fix problems in code, without dealing with the full complexity of a computing language at the same time. Here the reader learns how to trace the origin of an error to its roots and eradicate it.

The use of CSS shows the reader how to make maintenance of web pages simpler by eliminating duplicate code. The separation of document structure and document style achieved by this system shows a good example of modern thinking in computing. This discipline and this gentle introduction to the ideas of debugging, delimitation and grammatical discipline prepare the beginning programmer for what is ahead.

We encourage the reader as his study progresses to look at documentation on the web and to experiment with it. It is important experiment and to color outside the lines!

Chapters 3-6 deal with the core of a modern programming language in an object-based imperative setting. The notion of object as a smart, selfaware datum is introduced early. Abstraction and modularization are emphasized.

In chapter 3, basic features of Python are introduced and the importance of type as an establisher of context is emphasized. The various operators of Python are introduced, and their relationship to the types of the operands is emphasized. All of this is done using Pythons interactive mode. Then the

creation of free-standing programs is introduced.

Variables and their relationship to objects is discussed. The notion that variable are typeless labels is emphasized. Here the earliest ideas of object-oriented programming are introduced. You learn how to send an object a message via a variable to evoke some desired behavior. At the end of the chapter, the idea of a visible table of symbols is introduced and discussed in an extended example. The discussion of sequences at the end of the chapter is intended to accustom you to using objects to accomplish tasks.

In Chapter 4, we begin to study the *boss statements* of Python. All statements in Chapter 3 are *worker statements*. Worker statements are grammtically complete and syntactically correct sentences. They are self–contained commands. Boss statements are gramatically incomplete clauses. They require a group of zero or more worker statements to become complete commands. For example, the clause, "if age ¿ 21," is a grammatically incomplete subordinating clause. The command "pour a beer" is a grammatically complete (imperative) sentence. Glue them togeter and get, "if age ¿ 21, pour a beer." and you have a complete command. Boss and worker statements are present in all imperative programming languages, which includes Python. Chapter 4 addresses conditional execution and Python functions at the same time.

We reinforce the idea to the student that the the basic unit of Python code is the function. A function should do a single task in a focused way. Looping is deferred becaue we look at recursion as a repetition mechanism first.

The capabilities learned in chapter 3 become the grist for writing functions in Chapter 4. Early on, the idea of placing related functions in a single file and creating a module with it is demonstrated. The purpose of a namespace is then established, as a last name for functions.

Chapter 4 introduces an important principle of extreme programming: when developing modules, develop the documentation and test code for the functions first. This tells you that you have done enough thinking to know what the functions you are writing are to do. Then write code to test them. Running the test code causes your students (functions) to fail. As the functions are implemented correctly, they pass the tests. Your faith in the integrity of your module is thereby bolstered. In Chapter 4 we cheat and use the `if name == " main ":` construct to keep test code invisible when using the module but visible when the module is run directly. The test code is part of the modules documentation: it shows what vulnerabilities have been contemplated and tested by the developer of the module.

At the end of the chapter, we introduce the `random` module and demonstrate its functionality. Here the reader sees again that the purpose of modules is to create code that can be used to handle specific situations. The reader is encouraged to explore Pythons documentation and to see that there is a panoply of modules in Pythons libraries that accomplish a wide array of tasks. This chapter has two discussions of the call stack and the issue of scope. The first

discussion indicates the rules of scope. The second reinforces this discussion by showing how the call stack enforces the rules for visibility of symbols in Python. The chapter ends with yet another preparatory discussion revealing the importance of the intelligent use of objects via a discussion of the Python string type.

Chapter 6 begins with an exploration of recursion as a tool to perform repetition. We then do a detailed study of lists and their methods, and apply the tools of functional programming manipulate lists. All this is deliberately staged before the presentation of looping. Also, Python dictionaries are introduced in this chapter.

The `while` loop is explained as an indefinite loop and the `for` loop is show to be the tool for working with a collection: in this way, the `for` loop is a definite loop.

All of this brings us to a Turing-complete language which gives us all of the capabilities of a universal computer.

Chapter 7 contains a repository of useful tools. These can be introduced as appropriate. FileIO can be introduced when doing looping. Regular expressions can be introduced very early; these can be used with the UNIX `grep -e` command on a file. The os and os.path modules are natural accompaniments to FileIO. They too, can be introduced fairly early in the game.

Chapter 8 is an introduction to Python classes and the creation of custom Python objects. This is done via a case study involving playing cards.