Introduction

May 9, 2017

1 Introduction to Programming

Welcome to programming! The purpose of this notebook is to simply offer you an introduction to what programming is, as well as give you some clarifying words that I like to tell everybody just starting to program. Many people don't know (and don't need to know) exactly what goes on when they run a program on their computer, but some basic knowledge on the subject will put the rest of the course in context. When you know how the computer operates, you'll be able to adapt your way of thinking to write programs the computer can understand.

I've always thought of programming as being one part puzzle-solving, one part math, and one part arguing with a two year-old. Here's what I mean:

1.1 Programming is Puzzle-Solving

Computers are used to do things that we couldn't do, or that would take a very long time for us humans to do by hand. For example, if you wanted to calculate the average of two numbers, you might do it by hand, or at worst use a hand calculator. If you wanted to calculate the average of 1000 of numbers, you might use a spreadsheet. But if you wanted to calculate 1000 averages of 1000 different lists of numbers, programming would be the best tool for the job. The catch here is that there is only a small list of tasks that a computer can perform, and individually these tasks are not very useful. It's by stringing these different tasks together in different ways that we create complex programs. Below is a basic list of the things a computer can do:

- Store a value in a variable
- Perform basic arithmetic (+, -, *, /, %)
- Compare two values

That's basically it. Granted, the language you code in introduces a lot of structure around these three tasks, but pretty much everything can be boiled down to doing these things. When tasked with writing a computer program, beginners often simply sit down and try to start coding. This will sometimes work, but often a better method is to sit back and strategize. Think of how you would accomplish the task if you were doing it by hand, then break it down into tasks that the computer can actually perform. Then you're prepared to write a program that uses these basic tasks to accomplish something more complex.

1.2 Programming is Math

While a high level of math skill isn't required to program, a lot of problems can be drastically simplified by remembering some simple rules in math. The computer obeys order of operations (many people learn the acronym PEMDAS, or Please Excuse My Dear Aunt Sally). Also, most programming languages have a concept of functions, which take inputs and perform a task and/or produce an output. Here are some simple facts that you probably remember from math. I'm not listing them here because I think you've forgotten, but rather because these are useful tricks that come into play in a lot of programs:

- PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction)
- You can safely add 0 to a number and it won't change anything
- You can safely multiply a number by 1 and it won't change anything
- Before learning about decimals and fractions, we did long division and got two numbers: a quotient and a remainder. For example, doing 7 divided by 3 gave a quotient of 2 with remainder 1.
- If a number A can be divided by another number B, then dividing A and B will give a remainder of 0.
- Most of us start counting at 1, such that if we count from 1 to 10, we've said 10 numbers. However, when programming, it's often useful to start counting at 0. It's important to remember that if we count from 0 to 9, we've said 10 numbers.

Just keep these in the back of your mind as you work on problems and fix bugs.

1.3 Programming is Arguing

I don't have kids, but in my head, computers have a lot in common with small children. They often whine about things that shouldn't be issues, and can be obnoxiously literal when told to do something. But while nobody can predict what a two-year-old is going to do next, everything the computer does is deterministic. This means that the same conditions will *always* produce the same result. Computers will do *exactly* and *only* what you tell them to do, and will spit out errors if you ask them to do something they can't do. It's a harsh lesson to learn, but if the computer isn't doing what the user wants it to do, 99% of the time it's because of the user, not the computer. The computer is simply doing what the user told it to do.

One example is syntax. This is something that you'll constantly need to be aware of when programming. Syntax is essentially "grammar and punctuation" for computers. In an English class, a misspelling or missed apostrophe usually isn't the end of the world. The message almost always still gets across. This is not the case in computing. Even something as simple as a misplaced space or semicolon will cause an entire program to fail and spit out an error. However, don't be intimidated by this. Everyone makes syntax mistakes, and reducing them is simply a matter of practice. Just be warned that they will happen, and the results may be scary. Beginning programmers are often put off by the errors resulting from simple syntax errors that can be fixed in seconds. When you see an error in your program, before running for the hills, double check to make sure your syntax is correct.

1.4 What exactly is a programming language?

In essence, a computer is just a large set of switches that can be either "on" or "off." You've probably heard of binary language before: the language that is just 0s and 1s. The 0 signifies a switch

that is "off" and the 1 a switch that is "on." The computer really only understands commands that are given in this binary language. How then, are we able to communicate our intentions to the computer when we don't speak binary? Well, fortunately, somebody long ago thought this too, and developed programming language. Essentially, rather than typing a bunch of 0s and 1s into the computer, we can input commands that are "human-readable" and a separate program then translates what we say into something the computer understands. The first programming languages were still quite cryptic, but over time more advanced languages were developed that build upon these first languages, such that modern languages produce very readable code and hide a lot of the intricate details that many programmers shouldn't have to worry about. The language we will be using, Python, is one of these modern, easy-to-read languages. Just understand that there is more going on "under the hood" than what you can see.