1 Introduction

Welcome! Computer science is an exciting subject to study. By merely writing the correct words one can cause a machine to behave as one wishes. Is this image so different from that of wizards casting spells?! There are many forms of correct words: the different programming languages. In this course, Python will be the language used to illustrate the affect of words on the computer. Before going into depth on a particular problem and its solution, a few more remarks are necessary.

2 What is Computer Science?

One doesn’t often see the word ‘science’ as part of the the name of an academic area of study, and one may become suspicious of the description ‘computer science.’ Political science doesn’t seem particularly scientific; rather it seems to be some mixture of history and philosophy. Further, the classic sciences, such as biology and physics, do not have the word ‘science’ in their names.

The word ‘biology’ comes from the Greek where ‘bios’ (βίος) means ‘life’ and ‘logos’ (λόγος) means ‘speaking of’ or ‘study of’; hence the meaning is ‘the study of living things.’ Perhaps ‘computer science’ is written, instead of the more awkward ‘computer-ology’, to mean ‘the study of computers.’ Yet a computer scientist is more likely to be interested in building a computer than dissecting one, and where does programming fit in? The words ‘computer science’ don’t seem to be particularly helpful in clarifying what the subject is about; interestingly in several other languages, the name for this field is something like ‘informatics’ (for example, in French it’s informatique). How should science and related disciplines be understood, and where does computer science belong?

Science Knowledge concerning natural phenomena and their behavior.
Engineering Knowledge concerning the tools and techniques for building things.
Mathematics Knowledge of structure and form as determined by pure reason.

Computer science is an amalgam of engineering and mathematics. It is not science at all, since there is no natural phenomenon to observe. The engineering comes in both because computer scientists are obliged to know how computers are built, and because what they typically do is build computer programs. However, unlike bridges, computer programs can be understood as a thing of the mind, consisting of interacting algorithms. From this perspective, it is understood to be mathematics. Thus computer science is the study of algorithms and their realization in actual devices.

1. The word ‘algorithm’ is derived from the name of the medieval Persian mathematician Muhammad ibn Musa al-Kwarizmi (محمد بن موسى الخوارزمي) or Mohammed son of Moses the Khwarizmite. His math text introduced Hindu-Arabic numerals, and the methods of arithmetic using these numerals became known as the ‘algorism,’ a variation of al-Kwarizmi. This word evolved into its present form, and came to mean any effective procedure.
2. The study of algorithms is ancient. For example, there is documentation that an algorithm for approximating the values of square roots was known to Heron of Alexandria in the first century CE; further, it seems this algorithm was known to the Babylonians in about 1700 BCE (almost 4000 years ago).
3 What is a Computer?

A first attempt at defining the term ‘computer’ might involve pointing at one and mumbling, “That’s one; it’s a box that is connected to a screen and a keyboard.” Certainly the description as a box is insufficient; there are many objects that look like boxes. Next, one might be tempted to talk about what’s inside; for example, silicon chips. But it’s possible to build computers out of other components. Computers have been built from vacuum tubes, electro-mechanical relays, gears, and even tinker-toys! So perhaps it is worthwhile to focus on what a computer does rather than the materials used to construct it.

Today one can watch a DVD, listen to a CD, make a phone call, and type in a word-processor all on the same machine: the computer. Indeed, one can do just about anything with it; what makes the computer special is that it is an anything-machine. Of course, such a statement requires qualification. The computer can’t make breakfast. It’s not even of any use for playing a CD without speakers. The CD reader provides a digital input signal to the computer, and although sound is an analog signal, the hardware connected to the speakers expects a digital output signal from the computer. The computer transforms the digital input into the digital output that emulates the hardware device that a CD player would have had in the past. Similarly, the computer transforms the digital input into the digital output that emulates the hardware device that a DVD player would have had, and so on. The computer can emulate any device that transforms some digital input to digital output, as long as a suitable description of that device is provided; this description is a computer program.

4 What is a Computer Programming?

A computer program is a piece of text or texts that determines the behavior of the computer; it is the plan that turns our anything-machine into a particular machine that accomplishes what we want. How can a piece of text affect the behavior of a machine? Not that long ago (a span of a few decades) computers were programmed by flipping switches. Doing so would change the pattern of the magnetic orientation (or voltage levels) of certain components (named storage or memory) so that at run time certain electronic devices would be turned on and off appropriately so as to compute the desired result. These patterns are called machine-language or machine-code. Ultimately, it is the execution of machine-language instructions that makes things happen on the computer.

Fortunately, today it is not necessary to enter machine-language programs by flipping switches; instead, one simply types in the text of a high-level language using the keyboard. Note, though, that when one presses a key on the keyboard, one is flipping a switch. The signal from that switch

---

3. The philosopher Aristotle identified four types of explanations of why a thing is an X: (i) what it’s made of (ii) what its shape is (iii) how it was created (iv) what it’s for. Interestingly, often this last type is crucial concerning designed artifacts.

4. The word ‘digit’ comes from the Latin and means finger. Now digit refers to the ten numbers 0 through 9 that are used to write arbitrarily large numbers. More generally, digit can also refer to a finite sequence 0 through \( n \), where \( n \) is a positive natural number. Digital is the adjective form. A digital signal is a signal that can be represented as a sequence of generalized digits.

5. The word ‘analog’ comes from the Greek and is related to the word ‘analogy,’ which has to do with two things being similar such that one can help explain the other. Analog is the adjective form. Analog signal is a signal that represents the magnitude of one medium using the magnitude of another. For example, a sound of great intensity might be represented by a (relatively) large mass or by a (relatively) high voltage, and a sound of low intensity might be represented by a (relatively) small mass or by a (relatively) low voltage.
is converted into a binary\textsuperscript{6} number which is then stored in the computer memory. After typing an entire program, a sequence of binary numbers is in memory. This sequence cannot be directly executed by the computer. There are two basic approaches to execute that sequence. One approach is to translate the sequence into a sequence of machine-language instructions. The translator is another program called a \textit{compiler}. Another approach is to design a machine that can execute the sequence directly. It is not necessary to actually build the machine; it is enough to provide the computer with a description of this new machine. The description of this new machine is another program called an \textit{interpreter}. Python, for example, uses both approaches; a compiler translates Python code to ‘byte-code,’ and then an interpreter executes the byte-code.

While machine-language must conform to the hardware, one might wonder what a high-level language must look like. So far there are only four major programming paradigms (or ways of thinking about programming). (There are also a couple of lesser paradigms that won’t be mentioned here.) What follows is a list of each paradigm. For each one, an example will be given of the problem of putting away groceries.

**Imperative** As the name suggests, one gives orders to compute a particular result. To put the groceries away, one would say, “Open the refrigerator doors; put the milk on the top shelf; put the meat in the freezer; put the butter on the shelf in the door; close the refrigerator doors.”

**Functional** This name refers to mathematical functions\textsuperscript{7}; to write programs one defines functions, which then can be used define other functions, and so on. Evaluating an expression involving functions is used to compute a particular result. (Addition of two numbers is an example, where the addition operation would be the function in the expression.) To put the groceries away, one would say, “Apply the put-away function to a refrigerator and a list of food items; put-away then returns a new refrigerator that has the food items inside.”

**Object-Oriented** There are actually several different approaches within the object-oriented paradigm. What they have in common is a focus on objects, where an object is an abstract structure in the computer that represents a real-world object (such as a refrigerator). Often these objects interact by sending each other messages. To put the groceries away, one would create a refrigerator object, and then food objects corresponding to each food item where food objects know about the refrigerator and understand the message ‘put yourself away’; one would then say, “Ask every food object to ‘put yourself away’.”

**Declarative** This paradigm is quite different in that one only needs to assert or declare what the result should look like. To put the groceries away, one would simply assert, “A food item is put away with respect to a given refrigerator if the food item is in the refrigerator.” That example is very abstract; another example involves family relationships. One can assert that Jake is the father of Joe, that Joe is the father of Manny, and that X is the grandfather of Y if, for some Z, X is the father of Z and Z is the father of Y. One can then ask who Manny’s grandfather is, and be told the answer is Jake.

A language based entirely on a single paradigm is sometimes called a \textit{pure} language. Most programming languages are not pure. Python supports the first three of these paradigms.

---

\textsuperscript{6} An ordinary number is expressed using the digits 0 through 9. A binary number restricts the allowable digits to just 0 and 1. For the discussion here, it is enough to understand a binary number as simply a sequence of zeros and ones. Note that a binary number isn’t literally in memory; rather a binary number is an abstraction characterizing the state of the electronics. Typically, zero represents a low voltage, and one a high voltage.

\textsuperscript{7} Today a mathematical function is understood as any input-output pairing, with the restriction only that for a given input there must be exactly one output. Typically it is expressed as a rule. For example, the function that squares its input is written $f(x) = x^2$. 

A Very Brief History of Python

Python was created by Guido van Rossum in 1989 at CWI\textsuperscript{8} in the Netherlands. van Rossum's design was heavily influenced by his extensive experience with the ABC programming language and environment, which was developed at CWI as an education language. ABC, itself, was influenced by two languages that emerged in the late 1960s: ALGOL 68 and SETL — both high level, imperative, mathematically oriented languages.

\textsuperscript{8} Centrum Wiskunde & Informatica, or, in English, National Research Institute for Mathematics and Computer Science.