Concepts

Programming is Problem Solving

Programming is simply the process of creating a solution to a problem. A programmer is a little like a TV chef. The computer, in this case, is represented by the viewer. As the TV chef, you provide an exact instruction set for the viewer to follow in order to create the dish. Instead of using ingredients, detailed descriptions, and cooking implements, programmers use program syntax, comments, and the documentation for the coding language. Every program ends up being a recipe for the computer to follow.

A **program** is a coded behaviour that a computer can execute. Programs are the machines devised by programmers to solve problems. Often, these programs are meant to be applied to specific tasks. In these cases, the program is referred to as an **application**. This course will teach the basics of programming and allow you to create simple programs and applications. For now, however, you can think of a program as a series of instructions that the computer follows to carry out a task.

Algorithms

A program is executed by a computer in a step-by-step manner. When these individual steps are combined to carry out a specific task, they produce what is referred to as an **algorithm**. An algorithm is the step-by-step solution to a problem, but it is not necessarily the coded solution. For example, a baking recipe is an algorithm: the recipe is made up of a series of steps, and carrying out each step of the recipe results in a final product (hopefully the correct dish). There are a number of steps that you go through to bake cookies: laying out the dough on a pan, preheating the oven to 350 degrees, placing the pan in the oven, waiting 20 minutes, and pulling the pan out. Of course, this algorithm can be more or less specific. We might say “laying out the dough on a pan” as one of our steps, but the process might actually be more involved, such as laying down tinfoil, pulling apart the dough, and separating them evenly. The important part is that the directions are specific enough so that they can be followed exactly and the problem can always be solved.

Literalism

Before you start programming, you should understand **literalism**. Literalism is an adherence to the explicit meaning of a given text. Computers interpret programs *literally*. This means that computers will only do exactly what you tell them to.

If the computer falters in some way, it is a problem with the program rather than with the computer itself. This is known as a **bug** in the program. Much of programming is really a process of fixing the errors in the code. This process is known as **debugging**.

Often, the prevalence of bugs, and thereby literalism, frustrates new programmers. But, literalism is actually a benefit to programmers. Because of literalism, programmers can trust that the computer will always obtain the correct solution with the correct program; they can be sure that their programs will always function exactly as written. Once a program solves a problem, that program will solve the problem always and everywhere.

Syntax

Programs have to be written in a very specific way. The specific way code is written is known as the code's **syntax**. For example, every line of the C programming language has to end with a semicolon and every function in programming needs to end with parentheses. If the programmer forgets this, the program will no longer run. These requirements are known as **syntactical requirements**. Different programming languages have different syntactical requirements, just like different spoken languages. For example, in English, the subject of the sentence must always come before the verb, but in Latin, the subject can be placed anywhere in the sentence.

A bug can be a **syntax error**. This type of error is when the programmer mistypes some piece of code. For example, a variable or function might be named **`helloWorld`**, but the programmer could accidentally type **`helloworld`**. The coding language won't recognize it because **`helloworld`** is not actually defined, even though **`helloWorld`** is defined (programming languages are often case-sensitive). Syntax errors are the most common for new programmers.

Semantics

Programs are interpreted in a literal way. The interpretation is known as the program's **semantics**. Different pieces of code that perform that same exact task are called **semantically equivalent**. These pieces of code can even be in different languages, but their meaning is identical. Again, this is similar to spoken languages. For example, the English phrase "I came; I saw; I conquered" and the Latin phrase "Veni, vidi, vici" are semantically equivalent because they mean the same thing.

A bug can be a **semantic error**. This type of error is when the programmer codes something with the intention of doing one thing while the code actually means to do another. For example, a programmer might make a piece of code that divides a number by 2, but instead, the program multiplies the number by 2. In this case, the code is syntactically correct (because it completed the task given), but is semantically incorrect

[1] http://static.zerorobotics.mit.edu/docs/team-activities/ProgrammingPeanutButterAndJelly.pdf

because it did not function as intended. If the code does not behave correctly, but still runs, look out for semantic errors.

Exercises

1. Write an algorithm for the instructor to make a PB&J.[1] Have the instructor carry out the algorithm literally. Debug the algorithm until a successful algorithm is made.
2. Come up with a syntax to write the algorithm.
3. Write another algorithm that is semantically equivalent.
4. Using the syntax you made in question 2, translate your algorithm from question 3.

[1] http://static.zerorobotics.mit.edu/docs/team-activities/ProgrammingPeanutButterAndJelly.pdf