High-Level *vs.* Low-Level Languages

Language Hierarchy

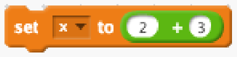
A wide range of languages exists, both natural (like English) and artificial (programming languages like Scratch). Ultimately, language is a tool for communicating an idea. In the case of computing, this communication may be human, machine, or anywhere in between.

At the *high* end of that spectrum lie the languages that are best for human communication. These languages are complex and use a high level of *abstraction* (a concept we'll explore in greater depth inUnit 3: Data Representation). They are symbolic and rely on strings of letters to form words that represent abstract concepts, ideas, actions, and objects. Our brains are optimized for this form of symbolic expression, so these languages are easy for us to read, write, and understand. Different programming languages offer different levels of abstraction. Examples can be found below. With all of its abstraction and inconsistent grammar and usage, natural languages are difficult to use in ways that will allow machines to understand our intent. With products like Apple's "Siri" interface or Google's "OK Google" voice search feature, the technology has made great strides in recent years, but there's still a long way to go before machines can truly understand natural languages.

Low-level languages, however, are optimized for machines. Computers are built using highly structured circuitry that responds to very logical and clear signals. Because low-level languages are much more concrete and straightforward, with limited vocabulary and very structured syntax, nothing is left to interpretation.

Machine languages are the most basic type of programming language. They represent the actual binary instructions issued to computer processors and are very difficult for humans to read. Moreover, they may vary from computer to computer! Believe it or not, in the beginning days of computer programming, *all* programs were written in binary machine language. Today, this is very rare.

The table below outlines this language hierarchy. Notice that the natural languages in the first row (e.g., English) are suited only for humans, and processor-specific languages in the last row (i.e., ones and zeroes, 10000010) are suited only for machines. The languages in between—high-level programming languages like Scratch, C++, Python—and low-level programming languages that tell the computer processor what to do (but are still somewhat readable by humans) bridge that gap, allowing humans and computers to communicate with each other.

Language	Characteristics	Example
Natural (English, Spanish, Chinese, Hindi, etc.)	Evolved naturally by entire societies through human communication. Potentially ambiguous. Easy for humans to read, write, and parse. Difficult for machines to parse.	*“Add 2 and 3 and assign the sum to a variable called ‘x’.”*
High-level Programming (Java, C++, Python, BASIC, Scratch, etc.)	Relatively easy for humans to read, write, and parse. Guaranteed to be unambiguous. Easy for humans to read, write, and parse. Easy for machines to parse.	`int x = 2 + 3;`  set x to 2 + 3
Low-level Programming (Assembly)	A direct translation of machine language using an abbreviated syntax. Guaranteed to be unambiguous. Less natural for humans, but still readable to the trained eye. Easy for machines to parse.	<pre> LD R0 2 LD R1 3 ADD R0 R1 ST R0 X</pre>
Low-level Programming (Machine)	Directly related to the hardware circuitry of the specific processor executing the code. Guaranteed to be unambiguous. Difficult for humans to read and write. Easy for machines to parse.	<pre>10000010 10010011 11000100 01001101</pre>

The high-level and low-level *programming languages* in the middle of the table are uniquely well-suited for computer scientists because they are abstract enough to feel natural and intuitive to humans, but basic and structured enough for machines to process with the level of precision that’s required to have them do what we want.

High-Level Programming Languages

Over the next few units, you’ll get the chance to work with two different high-level programming languages:*Scratch* and *Processing*. Scratch is a *visual programming language*, allowing you to drag-and-drop blocks to communicate with the computer without worrying about spelling, punctuation, etc., whereas Processing is a *textual programming language*.

One of the key features of most high-level programming languages is that, because of the abstraction they employ, people can program without needing to worry about (or even know about) the specific configuration or design of the computer's underlying circuitry. This allows the programmer to focus solely on the task of designing and coding a logical solution to whatever problem he/she might be working on.

Languages like Scratch and Processing are platform-independent, meaning that the programs you write will run on just about any modern computer, regardless of the model, version of the operating system (e.g., Windows, Mac OS, Linux, etc.), or maker of the hardware.

Low-Level Programming Languages

On the other end of the spectrum, low-level languages are entirely dependent on the underlying hardware. The language the computer uses is specific to the individual processor within the machine. That is, there is a direct correlation between the 1s and 0s of the binary code and the billions of microscopic, electronic switches embedded within the circuitry of the computer.

Fortunately, most programmers never need to actually work at such a low level or program directly with 1s and 0s. Instead, software development tools, such as the Scratch and Processing interfaces you will use in upcoming units, automatically generate the low-level, binary code that the processor requires. They do this by interpreting your high-level, abstract instructions into lower-level machine code through a process known as *compilation*, which we'll explore in the next section.