Computer Languages

Abstraction

Understanding the `1` s and `0` s of binary data is nearly impossible for any person. But if we can't read computer language, how are we going to properly instruct our computer what to do? The solution to this key problem of computer science lies in the core concept of **abstraction**. Abstraction is a way to manage the complexity of computer code and is used to help computer users ignore the many, small details that go into an operation so they can be more production.

In order to understand the concept of abstraction, picture yourself walking into a restaurant to buy a sandwich. Now imagine if the person at the counter tells you that you must go out, figure out what sandwich you, find the ingredients and make it yourself. This is where abstraction comes to save the day! Instead of making you do all the work, the chef already has the ingredients and the know how to make the sandwich up quick and easy.

The same process applies to computer languages and allows coders to forget about `1` s and `0` s and write computer instructions in a language more similar to our own. These computer languages follow a unique set of rules called the **syntax**. These rules, like the grammar and vocabulary of our own language, allow the computer to understand what we want it to do. Computers use a piece of software called a **compiler** to translate these computer languages into humanly-unintellible binary to be executed by the computer.

There are TONS of different computer languages out there for many different purposes. Python, Java, BASIC, and Ruby are a notable few of the many thousands of languages developed. We call computer languages that are closer to these `1` s and `0` s "low-level" languages and those closer to our own "high-level". These categories can sometimes be misleading and don't necessarily indicate a language to be more or less

sometimes be misleading and don't necessarily indicate a language to be more or less complex or useful. Low-level languages can give users more control whereas high-level languages can give users greater code simplicity and readability.

Remember that computers will only do exactly what you tell them to do and accomplishing seemingly simple tasks can take more operational consideration than initially believed. Take multiplication for example. From a computer science perspective, multiplication is best understood as a **derivative** operation of addition, meaning that multiplication can be understood in terms of the addition operation.

Let's try to figure out how to multiply using addition and write our solution in **pseudo-code**. Pseudo-code is a way to describe the steps of an operation simply and concisely without following the syntax rules of an established language. Programmers commonly use pseudo-code to describe a process or operation before the put it into syntactically correct code. Here is a solution written in my own kind of pseudo-code:

- multiply number1 and number2
- for number1 times, number2 is added to a total
- the total is returned as the answer

And voila! Those are all the steps needed to multiply a number by using addition only. Typing that code into a computer will not really cut it, so let's look at some code written in the **Python** language that will work.

```
def multiply(number1, number2):
    total = 0
    for idx in range(number1):
        total += number2
```

Do not fret if none of that makes any sense right now, it soon will! Before then, try thinking about other more complex operations that are composed of smaller parts and think about how you would get to constructing the former with the latter. Try writing some pseudo-code too!

P

Resources

- try out different languages here!