Which programming language is best?

If you have never programmed before or don't even know what is meant by "programming", those are the things I intend to cover with this post.

Programming in this context refers to Computer Programming, formally defined as:

"A process that leads from an original formulation of a computing problem to executable computer programs."

It might be tough to digest that if you're not entirely sure what a computer does at the lowest level. Everything a computer does originates with the central processing unit (CPU). The printed-circuit board (PCB) and circuitry around the CPU are there to provide an interface to peripheral devices that we as humans interact with: mice, keyboards, touch-screens, monitors, and more. In Figure 1 observe the CPU socket, which is where an actual CPU would be secured in place. Everything on that board connects to that CPU socket through the many layers of wires (called traces) that make up the PCB:

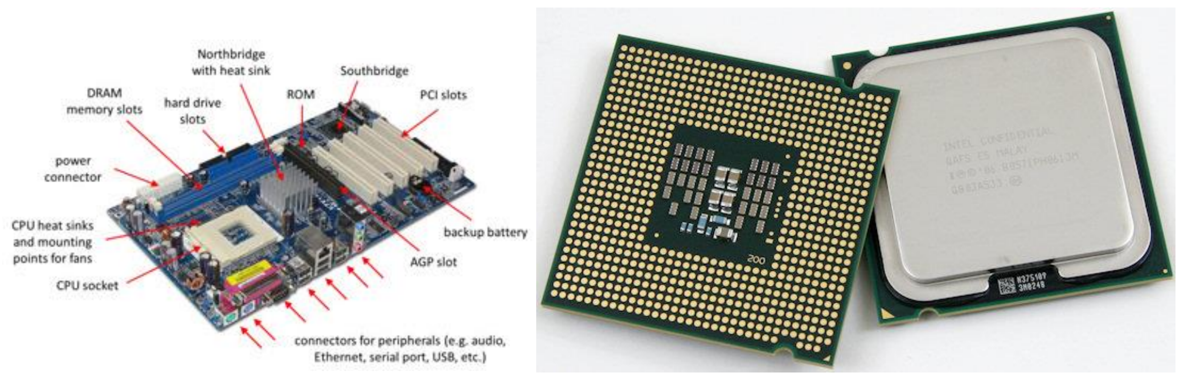

Northbridge with heat sink
Southbridge
DRAM memory slots
hard drive slots
ROM
PCI slots
power connector
CPU heat sinks and mounting points for fans
backup battery
AGP slot
CPU socket
connectors for peripherals (e.g. audio, Ethernet, serial port, USB, etc.)

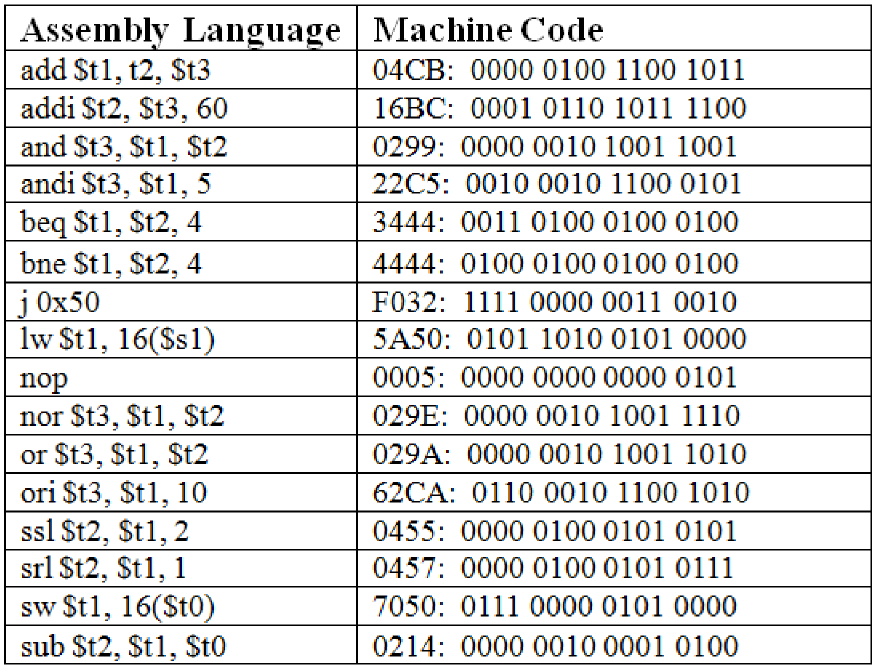
Figure 1: PCB and company on the left, CPU on the right

Many different types of things are controlled by and can be connected to computers, but does the CPU need to be prepared for each specific type of thing, and potentially things that haven't even been created yet? No, this is all made possible by international standardization and general organization of information, but a better answer to that is not within scope.

Know that a CPU can perform addition, subtraction, multiplication, logical comparisons, move data (integer and floating-point values in the form of binary numbers) and more, but only generic tasks that have nothing to do with the things as humans we expect our computers to do.

We have created a system in which we can use text to describe what we want a computer to do. First, we had to decide what language the computer should speak in terms of signal inputs and signal outputs. The way we organized that information is with instructions. Usually instructions are a fixed length of either 32 bits or 64 bits, where a single bit is just an on or off signal, denoted and detected by a difference in electrical potential (voltage). These binary numbers are interpreted by the CPU's circuitry to determine what to do with them, where the action item is one of the functions I described above.

We took this binary language and came up with a more human-readable version of it as seen in Figure 2:



Assembly Language	Machine Code
add $t1, t2, $t3	04CB: 0000 0100 1100 1011
addi $t2, $t3, 60	16BC: 0001 0110 1011 1100
and $t3, $t1, $t2	0299: 0000 0010 1001 1001
andi $t3, $t1, 5	22C5: 0010 0010 1100 0101
beq $t1, $t2, 4	3444: 0011 0100 0100 0100
bne $t1, $t2, 4	4444: 0100 0100 0100 0100
j 0x50	F032: 1111 0000 0011 0010
lw $t1, 16($s1)	5A50: 0101 1010 0101 0000
nop	0005: 0000 0000 0000 0101
nor $t3, $t1, $t2	029E: 0000 0010 1001 1110
or $t3, $t1, $t2	029A: 0000 0010 1001 1010
ori $t3, $t1, 10	62CA: 0110 0010 1100 1010
ssl $t2, $t1, 2	0455: 0000 0100 0101 0101
srl $t2, $t1, 1	0457: 0000 0100 0101 0111
sw $t1, 16($t0)	7050: 0111 0000 0101 0000
sub $t2, $t1, $t0	0214: 0000 0010 0001 0100

Figure 2: Translation of Machine Code (right) to Assembly (left)

The column on the left is somewhat understandable. "sub $t2, $t1, $t0" could have something to do with subtraction, where $t1 is subtracted from $t2 and stored in $t0, or something close to that.

The binary numbers on the right are virtually useless to us. We need a way to remember how we designed the computer to work using our verbs and language, and this is the first step to understanding computing: the translation from raw machine code to assembly language.

Also, I'd like to interject that "code" can refer to literally any line or snippet of something that's related to computing, in addition to the word's other meanings. Code has nothing to do with the readability or content of the text it has been attached to, it's just fast to say.

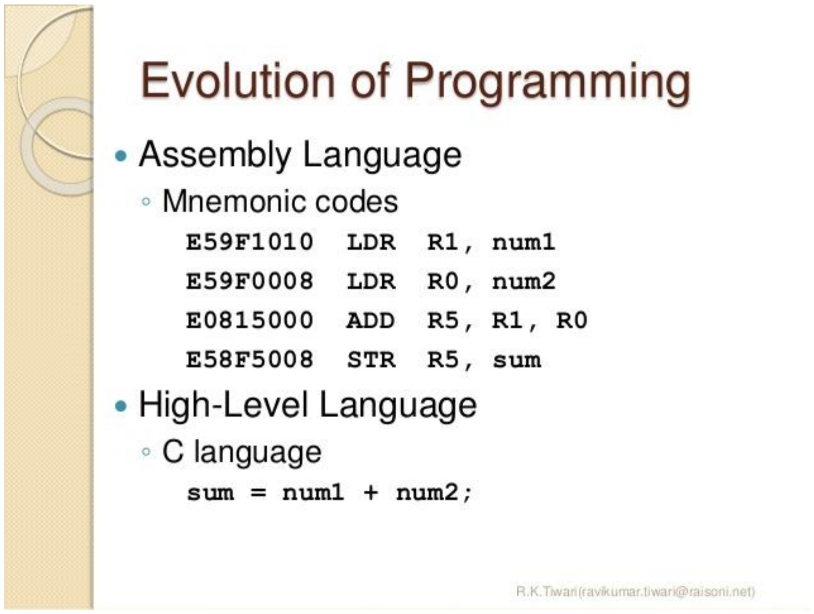
Where is code and data (essentially the same thing, code is data that is executed) stored in a computer? The answer is that there are dozens of different places that information is stored in a computer, and most of them serve different purposes. For now, we'll refer to the space for information and data as memory, and think of memory as one giant, walk-in closet with billions of hooks and hangars to store things permanently and temporarily. Assume the computer automatically takes care of knowing where everything is and where everything new should be placed, and that we have ways of putting things into memory for the computer to begin working with, but don't worry about knowing those details.

Now that we understand this system of instructions and computers processing information, we can begin talking about the different ways we have decided to translate the computer's language to our own.

An important concept to understand is layers of abstraction. Think of machine code as the lowest layer of abstraction: text is the literal numbers the computer works with. The assembly code is considered the next layer of abstraction, where each subsequent layer increases the extent of abstraction and (hopefully) decreases the time it would take to understand what the computer is doing.

If we try to increase abstraction from assembly, though, it won't be much of a direct translation, would it? Could it?

Remember that computers do a lot of math, and we as humans already have a language system for math. Maybe there's a way we can write mathematical equations as code and have the computer execute them?



Evolution of Programming

- Assembly Language
  - Mnemonic codes

```
E59F1010  LDR  R1, num1
E59F0008  LDR  R0, num2
E0815000  ADD  R5, R1, R0
E58F5008  STR  R5, sum
```

- High-Level Language
  - C language

```
sum = num1 + num2;
```

R.K.Tiwari(ravikumar.tiwari@raisoni.net)

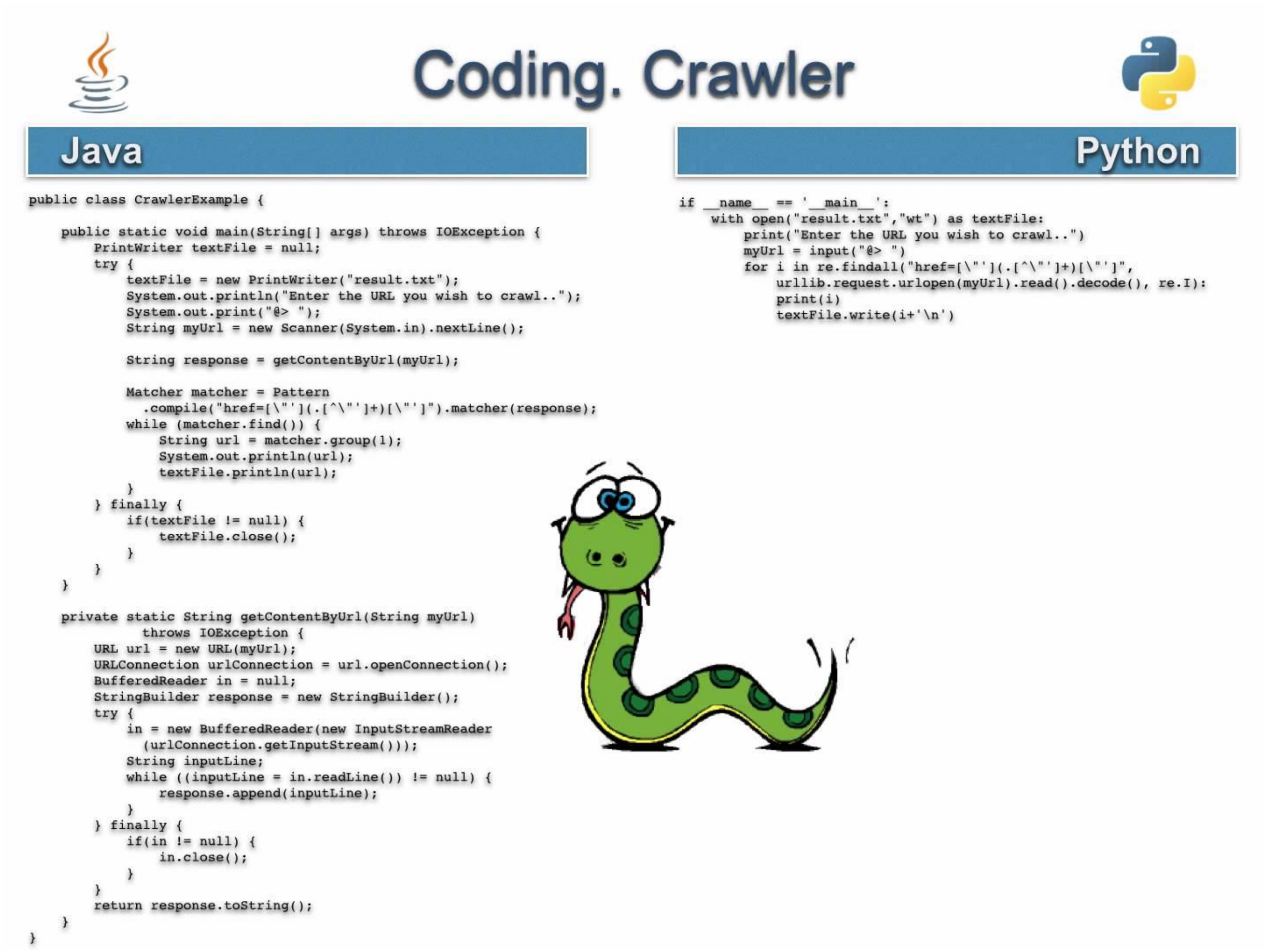
Figure 3: computer programming as a vehicle for doing basic math

In Figure 3 we begin to realize the need for a totally new and separate language from assembly. The Assembly Language example and High-Level Language example are meant to do the exact same thing: add two numbers and store the result in something called sum. We can see in the assembly that sum is literally stored in R5 (CPU register 5, registers are a container for information that we talked about previously). Hmm, we don't necessarily know that this is the case from the C code, though. Does that matter?

This is where we see the first instance of contention between two languages. In assembly, you're getting the whole story. Everything you see is exactly what the computer is doing, but, that has its flaws. Do we really need four lines of code to add two numbers? We do. If you look at that assembly code, there is a single line that performs the addition, but the rest of the lines are doing memory management types of things. One odd thing about this example that's worth pointing out is that to use names like num1. num2, and sum in assembly. This requires some entity to translate those names to RX (register X, potentially any register, and there are often 32 or less of these registers) or a memory address (billions of available memory addresses and locations) at some point in time. Is that legal? It is, but when writing this kind of assembly, you need a program that's already written to eventually translate it all back to RX, raw memory addresses, and eventually machine code. So naturally the types of programs that do those things need to be written in raw machine code.

These types of programs are called compilers, and provide us with a means to have C and assembly languages. They can take code written as text in one file and write the machine code to a new file. As the language becomes dissimilar to assembly, these compilers need to work harder and harder to do the translation.

I think at this point I have provided enough information for you to infer why it's an important question to talk about programming languages. We began by talking about computers at the hardware level and eventually landed on the means programming them. There's a lot to know when getting into computer programming, and the language you choose might not tell you the full story, which may mean that you never actually get to learn about assembly, machine code or the circuitry doing the dirty work. This might even be a good thing because you could create web applications and solve hard math problems without needing to go through a sequence of undergraduate Computer Engineering courses. We only looked at examples of C and assembly, but this post was becoming quite long so I decided it might be better to end more abruptly for your sake. I'll provide one last example to better illustrate the differences between languages:



```java
public class CrawlerExample {

    public static void main(String[] args) throws IOException {
        PrintWriter textFile = null;
        try {
            textFile = new PrintWriter("result.txt");
            System.out.println("Enter the URL you wish to crawl..");
            System.out.print("@> ");
            String myUrl = new Scanner(System.in).nextLine();

            String response = getContentByUrl(myUrl);

            Matcher matcher = Pattern
              .compile("href=[\"'](.[^\"']+)[\"']").matcher(response);
            while (matcher.find()) {
                String url = matcher.group(1);
                System.out.println(url);
                textFile.println(url);
            }
        } finally {
            if(textFile != null) {
                textFile.close();
            }
        }
    }

    private static String getContentByUrl(String myUrl)
              throws IOException {
        URL url = new URL(myUrl);
        URLConnection urlConnection = url.openConnection();
        BufferedReader in = null;
        StringBuilder response = new StringBuilder();
        try {
            in = new BufferedReader(new InputStreamReader
              (urlConnection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
        } finally {
            if(in != null) {
                in.close();
            }
        }
        return response.toString();
    }
}
```

```python
if __name__ == '__main__':
    with open("result.txt","wt") as textFile:
        print("Enter the URL you wish to crawl..")
        myUrl = input("@> ")
        for i in re.findall("href=[\"'](.[^\"']+)[\"']",
            urllib.request.urlopen(myUrl).read().decode(), re.I):
            print(i)
            textFile.write(i+'\n')
```

If you look at this example you can see how we try very hard to use English to program computers, and this makes things so much easier in the end. You run in to the problem of deciding what English to use and what types of formats should be followed, and those factors are essentially what inspires different programming languages and ideologies behind them.

The question of “Which programming language is best [to BEGIN with the learning]?” is the one I intend to tackle this semester. It should be interesting!

Vaughn Kottler

2/11/2017

EPD 397 with Mike Shapiro