What is Pseudo-coding?

An introduction to the art of turning a problem into code without even knowing how to code.

Scroll down...

Content | Resources (2) | Comments

We've spent a lot of time over the past few sections talking about how engineers solve problems. In the last section, we saw how this problem-solving approach is applied to the process of software

In this section, we'll start showing you how to design the solutions to problems without actually knowing how to code by using "Pseudocode".

You'll start by learning the basic logical structures of coding and applying them with pseudocode. Then we'll show you how to approach and solve complex development problems by using systems design principles and engineering best practices. Pseudocode is applied here as well, this time to model these systems instead of just individual procedures. By the end of the section, you'll have the engineering know-how to go out and be a truly effective developer.

So What is "Pseudocode"?

Pseudocode is basically just writing down the logic of your solution to a specific coding challenge using plain English. Or, as Wikipedia puts it:

> *Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading.*

Programming languages may seem complex, but ultimately, the vast majority of their components are completely interchangeable from one to the next. Whether a language is object-oriented or functional, it will still need to use conditional statements, functions, variables, loops and all manner of other common logical structures. Only the actual syntax for implementing them is different.

about whether you are using the "right" words. Once you've nailed the logic of coding, the rest is just filling in your pseudocode with the syntax of a particular programming language. In a lot of ways, pseudocode is similar to the mockups you created in the Design mini-course: it basically scaffolds the solution you'll be implementing later.

Let's say that I wanted to explain how a vending machine program worked to a general audience. I'd probably write it using pseudocode so you could understand it regardless of whether you are better at Python or Ruby or JavaScript or aren't even a programmer at all:

```
WHEN the user inputs money:
    IF the bill is too crumpled to read,
        provide an error message,
        and return the bill.
    ELSE,
        Add it to the balance
WHEN the user selects an item:
    IF they haven't put in enough money,
        ask for more.
    IF the item is out of stock,
        ask them to make a new selection.
    ELSE,
        determine change,
        dispense change,
        zero out the balance,
        and dispense the item.
WHEN the user hits the "cancel" button:
    IF the user has input money without making a purchase,
        return the money,
        and zero out the balance
```

As you can see, it's a pretty readable logical breakdown of what needs to happen with our vending machine. There's no "right" way to write

We also could have gone into deeper details or changed up some of the operations a bit, but the core logic has to be there somehow.

Pseudocoding for the Experienced Engineer

Identifying of the core logic of a problem is why working in pseudocode is so useful even after you've been developing for a long time. Most engineers who are trying to figure out a problem will first break out the pen and paper or a dry erase board and pseudocode (aka **Whiteboard**) the problem because it's easier to see all the moving parts that way and design a good solution. Others will write directly in their text editor and use the pseudocode as actual comments to guide their production of "real" code.

When approaching a new problem, you will often take an iterative approach: starting by modeling a broad overview of what needs to happen and then filling in each part that requires more detail.

For instance, in the problem above, you might simply start with:

```
Record the money being inputted;
Dispense the item if able to;
Handle any cancellation requests;
```

And you can probably see how each one of those gets fleshed out further. As you get more and more detailed, your pseudocode might start resembling real code, with variables and method names naturally coming out of it.

```
PROGRAM inputtedEnoughMoneyToBuyItem?:
    IF current_balance - item_cost > 0,
        return true
    ELSE,
        return false
    END
END
```

The above is sort of code and sort of pseudocode. How specific you get with your instructions is up to your preference and the problem at hand.

Check out this example of pseudocoding from Khan Academy.

Wrapping Up

In the next few lessons, we'll cover some of the logical tools you have at your disposal to break apart problems using pseudocode. They are the same logical tools that become real code! We'll lean towards presenting principles used in object-oriented programming languages like Ruby and Python, because that's what you'll be learning later on.

After mastering the concepts of pseudocode and the engineering principles it allows you to apply, you'll be more than ready when we dive into markup and coding in the Advanced Prep.
