The CS 1110 Declassified Survival Guide

Will Xiao

Spring 2019

1 Introduction

Welcome to CS 1110! We're happy that you decided to join us. Whether this is your first foray into the world of computer science, or you've done basic coding before, we hope you'll find the course as enjoyable as we do. Below, you will find tips on how to succeed and thrive in the upcoming weeks.

2 How to Study

Learning how to code is a lot like learning a new language. But instead of learning Spanish, French, Mandarin, etc, you're learning how to speak the language of a computer. Instead of writing essays to convey your ideas to other people, you write code that convey your ideas/commands to the computer, which the computer will then execute.

As such, you can treat CS much like you can a foreign language class. You should devote a little bit of time every day to learning and reviewing the material, outside of lecture/labs/assignments. If you don't speak a new language you're trying to learn for over a week, your skills are going to wither and you're going to forget a lot of the vocabulary/semantics. It's very much the same with programming. Try coming up with some examples of functions on your own using the topics you just learned in class, and try implementing them yourself. After you finish each weekly lab, take some time and sit down and think about *why* the code that you wrote down worked. Analyze it line by line. The better and more concisely you can explain it, the better grasp you have of the material. It's easy to sit in lecture and go "Ok, that makes sense", but it helps immensely if you can explain why everything makes sense on your own, without any guidance. As the course moves on to more abstract topics, being able to understand explain them clearly and concisely is key.

Also, going to office/consulting hours consistently is a great way to keep up with the course and make sure you definitely know what you're doing. Try to make it a habit (e.g. once a week or so). Even if you don't have questions yourself, it can be helpful to listen to other people's questions and the explanations they receive. This will especially come in handy in the days before the assignments

are due and office hours become incredibly packed- it becomes very difficult and frustrating to get help then. If you start assignments early, and come often to make sure you know how the concepts work, you will be able to get through the assignments and the course relatively stress-free.

3 How to Code Effectively

Coding is an art, much like writing. When putting together a paper, you can write a paragraph that gets the meaning across in a very roundabout way, or you can write a single, direct sentence that says the exact same thing. Writing code is very much the same. With that said, here are some tips.

3.1 How to Approach Writing Code

The best way to approach writing code is to break it down and reason about it in English. If you had to give a step by step process in English, describing how you would complete the task at hand given in the specification, how would you do it?[1] During this period, forget all about the fact that you're going to write code to implement the function- just reason about how you would go about it. If any of your descriptions seem very high-level/not easily implementable, continue further and break down that description into more directions/instructions that can be implemented easily. Once you have that done, then, and only then, should you start typing in the actual code. There is no point in trying to code without reasoning about the task first- it's like driving to a specific destination, but you have no idea how to get there. You'll probably drive around in circles/in the wrong direction for a while before you reach your destination. Same thing applies here.

As an example, say I wanted to write a function that takes in a string/text and returns the portion of that string that appears before the first letter 'e'. If I were to give a high level English description of this code, it might look something like:

1. Find the position of the first instance of 'e' in that string.
2. Get the part of the string before that position.
3. Return that substring.

Now that I've done that, it's much easier to actually write the code. It would look something like below. Note how I've made each separate step from above into its own line of code. When you first start learning how to code, separating each step into its own separate line can make things easier to keep track of and reason about.

[1] We call this step-by-step description of a process an **algorithm** in computer science.

```
def before_e(s):
    """
    Returns: The substring of s before the first 'e'.

    Parameter s: The string to get the substring from
    Precondition: s is a string with at least one 'e'.
    """
    e_pos = s.index('e')
    result = s[:e_pos]
    return result
```

As an aside, there's **always** more than one way to implement any given function/task- don't get discouraged if you can't figure out how to do it a specific way. This is especially important to keep in mind when you're trying to reason through a possible implementation of a function and your idea seems incredibly convoluted and long-winded. Maybe just try taking a break, and then coming back and looking at the problem from a different angle!

3.2 Making Code Elegant

In CS 1110, it doesn't matter how you implement your code. As long as it does what the specification says it should, you will receive full credit. That being said, the more simple and elegant your code is, the easier it will be for you to understand, and your graders will thank you for it (*hint hint*). If nothing else, elegant and pretty code is pleasing to look at.

3.2.1 Example 1: Boolean Logic

Let's start with a simple example, using boolean logic. Let's say I have a variable; call it *isPositive*. Let it store a bool representing whether a certain number is positive or not (if it is True, then the number is positive, and vice versa). If I wanted to do something based on the value of *isPositive*, I could say:

```
if isPositive == True:
    do something...
```

However, I could write this more cleanly, taking advantage of boolean logic, as:

```
if isPositive:
    do something...
```

This works is because if *isPositive* is True, then the expression $isPositive == True$ is always going to evaluate to True, so you're just replacing one True with another True. A similar analysis follows in the case where *isPositive* is False. Instead of saying $isPositive == False$, you could simply say *not isPositive*.

For a more concrete explanation of the difference between the two above statements, it is similar to the difference between saying *"If the proposition that the number is positive is true, then do something."* and *"If the number is positive, then do something."* In the end, both convey the same meaning. Which one is more pleasing to read and comprehend?

3.2.2 Example 2: Using Library Functions

As another example of how I could simplify code, suppose I wanted to find the larger of two numbers a and b. If I wrote the code for this myself, it might look something like:

```
def maximum(a, b):
    """
    Returns: The larger of the two numbers a and b.

    Parameter a: One of the numbers to compare.
    Precondition: a is a number (int or float).

    Parameter b: The other number to compare.
    Precondition: b is a number (int or float).
    """
    if a < b:
        return b
    else:
        return a
```

While the above code works, and it would still get the job done, I could write this much more concisely and clearly as:

```
max(a,b)
```

where *max* is the built-in Python function that returns larger of the two numbers a and b that you provide it.

See? So much cleaner and easy to understand. In general, if you're trying to do something with a built-in type (like strings, lists, etc), there's probably a library function or feature in Python that does that already. Take advantage of them- it'll make your code simpler and easier to read.

There are other ways to simplify and clean up your code besides understanding boolean logic or using the provided functions. If you go on to take CS 2110, you'll learn many other ways to make your code not only clean, but also efficient. I won't go into most of them here, but if you're interested, you're more than welcome to ask a staff member on how to improve your code- we'll be happy to help.

4 Extra Resources

If you're looking for extra resources past those given in class, here are some I recommend:

- **Python API:**
  https://docs.python.org/3.6/library/
  As annoying as it can be to read instruction manuals, this is one manual that you don't want to toss to the side immediately after opening the box. The Python API gives you descriptions of all the built-in features of

Python, including functions, modules, etc. Have this link handy- it'll tell you exactly what Python has to offer, and what you have to do yourself (this goes back to using built-in Python code to make your life easier/code more elegant)! While the table of contents may look scary, we won't be covering most of it in this class.

- **CodingBat:**
  https://codingbat.com/python
  Contains many different functions that you can work on to practice coding/use as drills. Their Python selection is somewhat limited, though. If you move over to the Java side, you'll see that there are more- I would recommend, if you're looking for more practice, to implement those functions in Python (such as in the Python Tutor) and then checking there by yourself (it gives you good practice writing test cases, too)!

- **Map of Computer Science:**
  https://www.youtube.com/watch?v=SzJ46YA_RaA
  While not explicitly related to the course material, this video provides a fascinating and engaging overview as to what computer science entails and to what is possible. I highly encourage you watch it, especially those of you who plan to pursue further CS courses after this semester.

5 Words of Warning

There are some things you should keep in mind while going through the course:

5.1 Consulting Hours

With just about 500 students in the course, office and consulting hours can potentially get incredibly busy, especially in the days leading up to assignment deadlines later on in the course. Start the assignment early, within the first few days of it being released, and beat the rush that will inevitably come two to three days before it's due. Waiting generally won't help too much, as the assignments are usually released after all the material you need to know to complete them has been gone over already in lecture.

5.2 StackOverflow

Online programming forums like StackOverflow can be both a blessing and a curse. It can be a helpful resource if you're stuck on a small detail and need to figure something out really quickly. However, things start getting a little hazy when you rely on it for bigger things, like how to implement larger functions/pieces of code. The main issue we have with the site is that it can lead to Academic Integrity violations if you're not careful. Sometimes, you'll be able to find code on the site, or GitHub, that is quite helpful for implementing something on one of the assignments. However, if you add it to your code

without citing, and some other groups do too, we will catch you (Moss, our AI detection program, is very good at noticing these unusual similarities). In general, if you're just looking to do something small, like 'How do I do X in Python?', it's fine. Don't try anything much larger though, like 'How do I implement this function?'

5.3 Codecademy

If any of you have tried to teach yourself programming in the past (possibly in preparation for this class), chances are you've used Codecademy or a similar resource to teach yourself the basics of coding. While these online platforms do teach you the basics of how to get off the ground with a language and get you up and running, they rarely (if ever) go deeper into the computer science theory and fundamentals of **why** things work the way they do. Codecademy teaches syntax. CS 1110 teaches computer science theory and good coding habits. If you do plan to use these resources to supplement your learning during the semester, please keep this in mind- always make sure to go back and review the theory on why things are that specific way.

Best of luck in the course! If you have any questions, feel free to talk to any of us on the course staff; we'll be happy to help!