Chris Piech, Mehran Sahami and Julie Zelenski Section #3

Code in Place April 30, 2021

Preparing for Section 3

This section is an opportunity for your students to internalize the interaction between variables and loops. Historically, variable scoping is particularly tricky for students to grasp, and this week’s section problems are targeted at clarifying this very important concept.

## **Learning Objectives**

In the last week, students have learned about using variables and loops, and were introduced to defining functions with parameters and return values. In this section, we will focus on the first of these concepts, as it is crucial for completing assignment 2. In the following week, students will have opportunities to practice defining and using functions, both in section and in assignment 3.

## **What can I assume students already know?**

1. **Loops with Variables (Last Friday Lecture)**

How to combine loops and variables. Boolean variable types. If/While/For.

1. **Parameters and Returns Part 1 (Monday Lecture)**

How to define functions which take parameters and give back returns. *Material in this lecture will not be covered in this week’s section.*

They **do not** know about: lists, dictionaries, tuples, returning multiple values.

## **Getting ready for section**

Send an email reminder about the section to your students 24 hours before your section. Make an announcement and tick the Send Email checkbox!

## **Recommended Section Plan**

### ***Social Time [5 minutes]***

While students are joining the meeting, take a few moments to “catchup”. In five minutes you probably can’t hear from each person one by one. A few options are: 1) have everyone answer the same question in chat. Or 2) take 3 minutes to tell your section a story. Could be related to programming, could be just something about you.

### ***Brief conceptual review [10 minutes]***

This week’s section focuses on variable mechanics in a loop. Work with your students to trace through problem 1, drawing attention to how the model of variables as luggage, or tiny boxes, extends to a variable that is reassigned to an arbitrary number of times. This problem is also a good opportunity to discuss good variable naming practices, and how a less-than-descriptive variable name might lead to subtle bugs.

***Running Total [10-15 minutes]***

The hardest concept to get about this problem is the idea that you need to declare the total variable outside the while loop. Work with your students to cement the idea that when a variable is defined outside the loop, any changes made to it *persist* upon completion of a particular iteration.

If your students solve this problem with relative ease, consider tackling the optional extension. Keeping track of a minimum and maximum value leads to some interesting initialization questions that are worth discussing as a group.

***FizzBuzz[15 minutes]***

Make a big point of being able to use the counter variable in the for loop. Print out the counter variable while you are solving the problem to start so that students can see it. Students often neglect to consider that the range function counts from 0 to n-1 whereas FizzBuzz should go from 1 until n. Additionally, students will likely have varying opinions on the best way to check each of the conditions, and in what order to do so. The solution makes use of constants and a conjunctive and check, but feel free to pursue alternative strategies that your students suggest.

**If you have time:**

While the goal of section this week is not to work on functions, your students might have questions about them if they are up to speed on lecture. If you have time left at the end of section, you might wish to discuss how functions could have been used to simplify the solutions to some of the earlier programs. For example, a get\_integer\_from\_user function would reduce repeated code in the section solutions.

## **Teaching Tips**

* Post to your Ed group early and often!
* You can pseudocode in whatever way makes sense to you (whiteboarding, typing into a Google doc, etc.) as long as the students are the ones coming up with the steps! One format we recommend is writing your pseudocode as comments in the code.
* Avoid using terms like:
  + “This is simple/easy.”
  + “You should already know how to do this.”
* Let students make mistakes and roll with them! You can turn these bugs into learning moments by going back and saying, “What might have gone wrong?”/  
  “Where do you think the bug might be?”
* Make sure to run the code frequently to see if things are working as expected. This instills good testing habits for students.
* Value different solutions - encourage students to think about other ways to solve the problem if you finish one solution early, and discuss the potential pros and cons of each!

## **Code in Place conventions to keep in mind (again, not updated yet)**

* All code must be written inside a **main()** function that gets called under:  
   **if \_\_name\_\_ == “\_\_main\_\_”:**   
  We want to avoid having any code (besides constants and imports) outside functions.
* We don’t explain what the if statement above means until later in the course. If students ask about it, just mention that it’s necessary for the computer to run the program, but they don’t need to worry about it for now!
* In Code in Place, we make a clear distinction about “functionality” and “style.” Style has to do specifically with code readability and reusability. In the class, there are conventions that we emphasize and that we do not want you to deviate from:
  + We follow Python’s [PEP 8](https://www.python.org/dev/peps/pep-0008/) style. This means that variables and function names use “snake case” (all lowercase letters with underscores for spaces). Constants (which will be taught later) will be in all capital letters. **Please do not use camel case (e.g. myVar) or any other type of naming conventions.** If you’re not familiar with Python or PEP 8, take some time to skim the online guidelines.

Your function and variable names should be **descriptive but concise**. Avoid names like **var1**, **var2**, **foo()**, **bar()**, and **function\_that\_has\_very\_long\_name()**