Chris Piech and Mehran Sahami Section #1

Code in Place April 14, 2021

Preparing for Your First Section

This is going to be the first week of coding for your students. What a beautiful thing :-).

## Hospital building Karel

[Here](https://edstem.org/us/courses/9999/lessons/12453/slides/60985) is the section handout and code the students will see!

We are going to get Karel to build simple hospitals. The main point of this first section, however, is more to build a bond with your students. Love of Karel is a great place to start.

We hope that you get to know your section, that they get to know each other, and that they understand why section is such a cool and important part of Code in Place.

## Learning Objectives

Students are learning about control flow and the very basics of coding. Specifically:

1. How to define helper functions
2. While loops
3. If statements

You can’t teach all of that in 5 minutes. Instead your main job is to give students a little practice and give them a chance to form a learning community.

## What can I assume students already know?

Before joining the class, students finished some problems that used basic for loops. They are also supposed to have read the [Karel reader](https://compedu.stanford.edu/karel-reader/docs/python/en/intro.html) and have watched videos which cover:

1. **Intro to Karel (Lecture 1)**

Course logistics, basic Karel commands, and running programs in Karel

1. **Functions (Lecture 1)**

In class we teach Karel how to **turn\_right()**.

1. **Control flow in Karel (Lecture 2)**

Loops (**while** and **for i in range(n)**) and basic **if** statements

Are you new to teaching Karel? You would probably benefit from reading the lecture slides or even watching the lectures! The first lectures are really fun. Having said that, to save you time we are going to put all the crucial concepts here.

If a student asks a question about a concept that has not been covered yet (like variables), let them know that the concept will be covered later in the class, and that they can send a private message to you on Ed if they have more questions. Resist the urge to go into the weeds on topics we haven’t covered yet! We will get there :-)

## Recommended Section Plan

### Getting to know your section [15 minutes]

Feel free to spend a good amount of time learning names and establishing norms for your section. We encourage you to pick an icebreaker to build community among your students! Here’s a [list of icebreaker ideas](https://docs.google.com/document/d/1rSjLi-80LMQf7wQ_mNmB78E0iRdpRa4TiwWAcqbDYlI/preview) that are good for Ohyay and help avoid revealing any personally identifying information about yourselves or your students.

Remind them that you are there to help answer their questions about section material and that they should ask all questions through Ed. Make sure they understand that they will have one Ed for the entire class (where all questions about Python and the overall course should go) and one Ed for your section. But if students begin to ask many detailed logistical questions, you can have them post on Ed so that you have enough time for Karel.

### Brief conceptual review [5 minutes]

**This week’s section focuses on decomposition, for loops, and while loops** and will help prepare students to get started on Assignment 1. In particular, we want to solidify the basics of loops for control flow and thinking through how to break problems down into smaller parts.

Although students are expected to have watched Wednesday’s lecture, we recommend briefly reviewing at least the concept of while vs. for loops (when would you use one vs. the other?), in case some students are not caught up on videos. Since they will have completed Assignment 0 and have seen Karel there, you should spend more time reviewing control flow topics rather than reviewing basic Karel commands.

Additionally, reviewing the motivation and technique of decomposition is important; students might have questions about what sorts of tasks should be decomposed, and this is a good time to help them build that intuition.

### Teaching Hospital Karel [20 minutes]

See the step-by-step instructions below.

## Step-by-Step, Teaching Hospital Karel

1. Start with the assumption that **build\_hospital()** function exists. How do you solve the problem? This is the most interesting part of the section.
   1. A common bug in students might make is excluding the second if statement that prevents Karel from crashing into the wall at the end of the program. If they do, don’t point it out before coding! Let them make the error and then talk through why it happens. (This is an example of an off-by-one bug, or a fencepost error!)
   2. When explaining the second if statement inside the while loop, it’s important to emphasize that the while loop condition only gets tested at the end of the loop. A common misconception about while loops that students often have is that the boolean condition is checked constantly.
   3. An alternative way to solve this fencepost error (although slightly more clunky) is to duplicate the **if beepers\_present()** block above the while loop. If students solve the off-by-one bug this way, that’s also fine! You can show them the solution code and explain why both solutions do the same thing.
2. Code the while loop. Use **put\_beeper()** in place of the body for **build\_hospital()**. Run!
3. Implement **build\_hospital()**.

## Teaching Tips

* When planning out programs, Chris and Mehran emphasize thinking about the pre- and post-conditions for programs and functions. **Ask your students what the pre- and post-conditions for each function will be.**
* Make sure your students are the ones coming up with the solutions (avoid having the solution code already written and open for them to see).
* 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

* 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()**.

## Karel Appendix

|  |  |  |
| --- | --- | --- |
| **Base Karel commands:**  **move()**  **turn\_left()**  **put\_beeper()**  **pick\_beeper()** | | **Conditions:**  if *condition*:  *code run if condition passes*  if *condition*:  *code block for "yes"*  else:  *code block for "no"* |
|
| **Karel program structures:**  # Comments can be included in any part  # of a program. They start with a #  # and include the rest of the line.  **def main() :**  *code to execute*  *declarations of other functions* | | **Loops:**  for i in range(*count*):  *code to repeat*  while *condition*:  *code to repeat* |
|
| **Names of the conditions:** | | **Function Declaration:**  def *name*():  *body of the function* |
| **front\_is\_clear()**  **beepers\_present()**  **beepers\_in\_bag()**  **left\_is\_clear()**  **right\_is\_clear()**  **facing\_north()**  **facing\_south()**  **facing\_east()**  **facing\_west()** | **front\_is\_blocked()**  **no\_beepers\_present()**  **no\_beepers\_in\_bag()**  **left\_is\_blocked()**  **right\_is\_blocked()**  **not\_facing\_north()**  **not\_facing\_south()**  **not\_facing\_east()**  **not\_facing\_west()** |
| **Additional commands:**  **random(***p***)**  **paint\_corner(***color***)** |