Clarifications for Question 9

- Before any component machine is stepped the first time, it must be reset.
- You can assume that each machine in the sequence, after it is reset, will have to be stepped at least once before it is done.
- Be sure that your `step` method returns an output.
- The `done` method may be called many times, and shouldn’t change the state of the system (that is, it shouldn’t advance the counter or call `step` on any component machines).
- When you call `run` on a text sequence machine, don’t worry if the very first character is repeated twice, or if there is an extra `None` at the end. (You should be able to trace through the code, though, and understand why it’s happening).
- Write your `done` method first; think about how you’re going to know when the machine as a whole is done.
- Here’s a way to think about what has to happen in your `step` method. Let `c` be the counter that is keeping track of which component machine is being executed, and `m[c]` be the `c`th component machine. We want to be sure that, if the whole machine isn’t yet done, that we make exactly one call to `step` on a component machine.
  - If the composite machine is done, just return
  - Else, if `m[c]` isn’t done, then step it.
  - Else (`m[c]` is done)
    - Increment `c`
    - If `m[c]` is a machine in our list (c isn’t too big)
      - Reset `m[c]`
      - Step `m[c]`
    - Return the current output

Clarifications for Question 10, 11

- In the example code for `GoForwardUntilXLimit`, the line `def self.currentOutput()`: should be `def currentOutput(self)`: 
- These behaviors are given sensor readings as input and should generate actions as outputs. They shouldn’t call motor command. And the only place you should need to call `collectSensors()` or `pose()` is in the `reset` method.

Clarifications for Question 12

- Put your `TBDrive` and `TBTurn` classes in the brain file.
- Be sure to load a simulator before you load your brain.