REVIEW FEEDBACK

Harry Turnbull 21/05

21 May 2020 / 12:00 PM / Reviewer: Katherine James

Steady - You credibly demonstrated this in the session.
Improving - You did not credibly demonstrate this yet.

GENERAL FEEDBACK

Feedback: Well done on a really excellent review! You are nice and steady across all the course goals and have a solid process. Just watch out with your test progression for special cases vs general cases. A particular congratulations on the improvement to your process of consistently and iteratively modifying your existing code, after the addition of each new test.

I CAN TDD ANYTHING – Steady

Feedback: Testing approach: Your tests were based on overall input and output behaviours, so were nicely decouple from your code.

Test progression:
This was pretty much the only thing I picked up to improve on. Having equal limits is actually a special case of the general case, so limits=1,1 isn't as simple as to say limits =1,2. This also created a bit of a problem, as because high=low you were able to perform a refactor to just return [low] and were only forced to differentiate between these cases later on in your code as a result of another test. If you could adjust these tests to using different limits, then these tests would have been great ones to start with.

Other than this and with the addition of some examples with different limits, your test progression was really nice. You started with a single value in the range, then below the range. I would have liked to have seen the addition of a test like the one on line 19 to similarly check for a frequency above the range, before moving onto examples with multiple frequencies. This means you

sort out all the interactions between things at the same level at the same time, and prevent things like using ternarys before you are ready for them.

You mentioned that you find it difficult to plan your tests because you don't know what the state of the code will be. That's completely fine, you always want to find a new test that breaks the assumptions of your code.

Red-green-refactor (RGR) cycle:
You got your simple tests to pass really nicely though simple if-statements and built-up your code incrementally. You stuck to your cycle well.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You were highly familiar with Ruby syntax and programming. You used map straight away, which was a really good choice for this task as it is nice and efficient and demonstrated familiarity with if-else statements.

You nailed the task quickly, so I gave you an extended task. You handled this really nicely, showing familiarity with Ruby's time module.

I CAN DEBUG ANYTHING – Steady

Feedback: You had a really logical and methodical debugging process, where you iteratively checked one thing at a time until you located your bug. You could find the most important part of the error message in the stack trace.

I CAN MODEL ANYTHING – Steady

Feedback: You modelled your solution in a single method which I felt was a nice and simple implementation and provided a good place to start. Note that for best practices, method names should be 'actionable', so should contain a verb.

I CAN REFACTOR ANYTHING –Steady

Feedback: You refactored your initial if-statements to just return low. Given the test cases you had, this was entirely valid. You later refactored your new if-statements to a ternary operator. This was also fine given your tests, but as you hadn't accounted for one of the simple cases yet, differentiating frequencies above the range, you had to undo this later.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: Your process was really methodical throughout; good information gathering with to the point questions, you used three panes for easy reference between your IO table, tests and code, you ran Rspec nice and regularly to check that error messages were changing, used nice conventions for git commit messages and double-checked the requirements to make sure you had met them all.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You gathered the technical requirements of the program through a couple of excellent and to the point questions. You started off in the logical place, verifying what a band pass filter does and then asked a number of follow up questions to clarify different aspects of the program operation. These were about the value to which the frequencies outside the range should be adjusted to, the soundwave input type to the program and the data type of the frequencies inside it. You probed into how the input might vary, with questions about the length, negative numbers and whether the array might contain any strange values. This brought up a discussion of the edge case for corrupted input. You confirmed the output format, showed how to call the method with the three arguments, asked about naming preferences and then created an input-output (IO) table. It might have been nice if you'd tried to think of a few more edge cases, such as if the limits were swapped.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You committed your readme, your updated readme with the IO table, and thereafter on completing both the red and green steps of your RGR cycle. You sometimes missed the red commit till you were ready to do the green one, but the green one is the most important so this was okay, as the last working

version of your code is always backed up then so that you can revert back to it if something should go badly wrong during the next step of your cycle.

I was pleased that you had your test suite properly decoupled from your implementation by making sure the tests were based solely on acceptance criteria, and not reliant on the current implementation. This makes changes to the code much easier.

I CAN JUSTIFY THE WAY I WORK – Strong

Feedback: You verbalised your process really nicely, so it was easy to follow your process and what you were testing. You let me know when you expected tests to fail, what errors you were getting and how you were going to address the error, and when they passed.

You provided really nice reasoning for changing your tests, that the outputs for the ones you had are all the same, so it wouldn't drive any development in your code. You justified the addition of new tests really nicely.