REVIEW FEEDBACK

Chris Cooney 27/08

27 August 2020 / 01:00 PM / Reviewer: Katherine James

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

GENERAL FEEDBACK

Feedback: There was a notable improvement in this review over the first one, where you got to the point where you were testing acceptance criteria much sooner than last time. You could still refine your process further here, by writing tests based on the final input and output from the word go, starting with the limits supplied in the case today and testing explicitly for the case where they were removed later on, or hard-coding magic numbers in until you introduced a test where they would be supplied.

I CAN TDD ANYTHING – Steady

Feedback: In today's review, your TDD drove nice and incremental changes to your code. There was much less modifying of tests and you got to testing acceptance criteria sooner.

However, there could still be an improvement in how you handled the default values as initially, you were moving between your tests and code a lot and didn't test the default values in a very clear way - not using acceptance criteria based tests.

The way you handled this was as to initially test that given integers, you could return these integers, then upgraded these tests and test three cases in which you converted the default argument to a string, and returned a string interpolated result with a frequency and the default limits. The purpose of this was to check that the default limits were working correctly. Thereafter, you took a step towards the acceptance criteria and given an integer of 10,

return 40 (the default lower limit) and given a value of 2000, return 1000 (default upper limit). You used if-statements to make these test pass, then upgraded the inputs to be arrays (outputs still integers) and then introduced a new test and moved all tests to acceptance criteria.

To refine this early part of your TDD process I would recommend that you write tests based on acceptance criteria from the word go. Eg: Given [10] return [40] and then either use a hard-coded ‘magic number’ for this case, or start by explicitly supplying the limits (which would probably make the most sense), ie. Given [10],40,1000 return [40] . This would mean that there was greater clarity in terms of identifying what functionality each test introduced to the code. If you took the latter approach, you could then later on explicitly test for a case with the limit values not supplied and handle the default values this way clearly, completely avoiding tests not based on acceptance criteria. You could also make the single-valued tests pass while based on acceptance criteria simply by accessing the array at index zero in the if-statement check.

Your final test progression was very good. You started off with single frequencies in the array, both in the range and below the range and from here incremented the complexity to handle multiple frequencies. It would have been good to write a third test simple single-valued test, for the in-range case. This would allow you to complete the logic for ‘how to handle a single frequency of any input case’ before moving onto ‘how to apply the logic I have to multiple frequencies’.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You developed a good solution and were nicely familiar with the syntax necessary for doing so. You demonstrated familiarity with using a for loop, if-statements and push for array manipulation. You were also familiar with type checking and setting default values. Using your knowledge of syntax, you fluently built up a solution.

I CAN DEBUG ANYTHING – Steady

Feedback: Your debugging process seems to be nice and steady. You were familiar with the common errors from early failing tests and could identify the different parts of the error message in the stack trace. You used the output of failing tests to determine how your code needed to change and also used console.log to get some visibility into your code.

I CAN MODEL ANYTHING – Steady

Feedback: You modelled your solution in a single function which I felt was a nice and simple implementation and provided a good place to start.

You named your function ‘filter’. This was a good, actionable name, although it wasn’t extremely descriptive. You could use lowerCamelCse to make the name a bit longer and more descriptive. Eg; filterFrequencies. This is just a suggestion, as filter is perfectly acceptable.

I CAN REFACTOR ANYTHING –Nothing here

Feedback: You considered refactoring your set of if-statements to a switch statement but opted to step back from this. A switch statement is often a very nice upgrade from if-else, particularly if there are 4 or more branches to your if-else.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: Your process was notably more methodical today as you ran a much tighter red-green-refactor (RGR) cycle. You were still starting with tests not based on acceptance criteria and moving towards this, with the purpose of ensuring that your default values were working properly. It would be more methodical to start off right from the beginning with tests based on acceptance criteria, explicitly supplying the limits at first and then, later on, testing explicitly for a case without limits supplied, to allow the introduction of the defaults.

I USE AN AGILE DEVELOPMENT PROCESS – Steady

Feedback: You conducted a great info-gathering session. You reified the main requirements nicely as well as finer details, ensuring you were on the same page as the client. You started off by establishing how the frequencies would be supplied and followed up by asking about the values of the allowed range. This brought up a discussion about arguments and also about default values. You also considered if there would be any strange values in the input as edge cases. As one point of refinement, it is quite important to clarify if error messages should be returned or if error messages should be raised. You then confirmed your understanding of the program requirements thus far by giving a succinct overview. You then confirmed how values which fell outside the range should be treated.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: By the end of the session, the changeability of your code was good as although your tests were initially coupled to your code after you had assured yourself that the default arguments were working, your tests were then based on the final input and output of the system and thus were nicely decoupled from the code, such that the implementation could change without breaking the tests.

You also committed after tests passed and supplied a commit message. These commit messages documented the changes you made to your code and made it possible to roll back to previous versions if necessary. You also chose variable names such that it was clear what they represented, which made your code easy to read and thus change.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You vocalised your process so it was clear what you were doing and why as you developed your code. You highlighted the reasoning behind your test progression and pitched your comments at a good level, such that it was easy to follow your progress. You let me know what error messages you got, when

these error messages where what you expected and highlighted what sub-task you were tackling at present as a step towards getting a test to pass. You let me know what you had achieved at notable points and what still needed to be done.