REVIEW FEEDBACK

David McGregor 16/04

16 April 2020 / 11:00 AM / Reviewer:

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

GENERAL FEEDBACK

Feedback: Thanks for coming in for another review! It's been great to see how you have made progress over the reviews - you are now steady or strong in all course goals, congratulations! It was also great to see how you coped with an extension to the task which made you work with a module, benchmark, that you hadn't worked with before. This really highlighted your methodical approach and processes, and I was suitably impressed. In the individual response boxes for each goal, I have given an overview of your process and have tried to highlight even further ways to make your process even stronger.

I CAN TDD ANYTHING – Strong

Feedback: I was impressed with your TDD process - you followed a Red-Green-Refactor cycle well, stuck to the very good test progression you had laid out in your IO table and didn't get ahead of yourself at any point. You allowed the tests to drive the development and so you spent very little time having to design the program or debug. This was a demonstration of the benefits of TDD. I was pleased to see that you identified the three core cases (within bound, above and below bound) and based your initial tests on simple cases of these. Your second test actually was a "duplicate" of the first in that it tested the same case, but as long as you don't get into the habit of creating a lot of duplicate tests (in which case this would be bad, as they do not drive development in the code) then this one on its own is fine.

It was good (and right) that the code you wrote was at just the right level for the test you had written. I also liked how you handled the single element

array by first simply accessing the first element. You dealt with your edge cases very nicely and your tests covered all behaviours of the program. Your approach made you able to produce an excellent solution very rapidly.

I CAN PROGRAM FLUENTLY – Strong

Feedback: You moved through the programming part of the task without hesitation, confident in the use standard Ruby syntax and with map for collection manipulation. This was a great choice, it might be interesting for you to have a look at how long the less efficient each loop may have taken with your benchmark testing. You developed an excellent solution, with lovely clean code. It was good to see you making use of the Ruby docs to identify a module to use for timing when I added this to the task near the end. You coped with using a module you hadn’t used before very well.

I CAN DEBUG ANYTHING – Strong

Feedback:

I CAN MODEL ANYTHING – Steady

Feedback: You followed naming convention best-practices and named your class as a noun and your method as a verb.

You modelled your solution after a single method encapsulated in a class, this was a simple enough place to start and also provided a great foundation for expanding complexity later on in the software's life-cycle.

I CAN REFACTOR ANYTHING –Steady

Feedback: You were aware of having a separate refactor step and made sure to refactor whenever appropriate, such as after you very first test to just return the input. This was an appropriate refactor as it allowed you to generalise your code and from this point naturally evolve it to check the

input value against the thresholds. You also refactored nicely when dealing with edge cases, moving away from a hard-coded string to any string, then after implementing code for checking other types, refactored to check for anything that wasn't an int in the input. I also particularly liked how you started with the code checking just the first element and then expanded it to checking throughout the array.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Strong

Feedback: You have a nice methodical approach to your development. You moved from requirements gathering to developing a good set of inputs and outputs that drove your testing. You clearly put a lot of thought into this step when developing your IO table. You were methodical throughout your process, particularly your debugging.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You asked for an example of the expected input, which you used to determine how the soundwave was modelled, and whether there was a single or many inputs, and what these all represented. You then asked about the overall program function to determine what the output should be. You can add further refinements to your process here by specifically asking about the output format, clients can be strange and want the output in a non-logical format, so by specifically clarifying this you eliminate this possibility - although this was effectively taken care of in your IO table.

You asked about edge cases such as the order of the thresholds and what to do with inputs of other types. You made an excellent IO table, in which you laid out your test cases and provided an opportunity for the client to identify any discrepancies between your views on the program.

When the input is integer based, these are some other common possible edge cases to consider: are there any default values, is there a maximum or minimum value these can take, can they be negative. There could of course and are likely to be more, but these are usually domain-specific.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: Your code has good changeability. Your tests were nicely decoupled from your code, and you made regular use of Git. Your choice of modelling the solution as a class with a method also provides easy expandability.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: I really felt that I could follow what you were doing well. Your vocalisation was good in this regard, allowing me to follow your process and understand the decisions you made. To make your vocalisation even more purposeful, you can let the reviewer know what the case is for the test you have just written, ie: this test checks for a single value below the threshold. You did this at times, but at other times it would also have been beneficial.