REVIEW FEEDBACK

Natasha McIntyre 26/03

26 March 2021 / 02:00 PM / Reviewer: Pierre Roodman

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

GENERAL FEEDBACK

Feedback: There are many improvements to your process once again and it is good to see that you take feedback and implement it in your process. The main thing that I would suggest that you focus on is trying to identify potential refactors on refactor phases in order to develop clean code throughout the development process. Very well done.

I CAN TDD ANYTHING – Steady

Feedback: Your test progression is really good as you have maintained the good habits that I discussed in your previous written review and also made the refinement of not introducing tests that would automatically pass in order for every red test to bring a transformation to the algorithm in an iterative fashion.

You have generally followed the RGR cycle, but there were phases that presented opportunities to refactor which you did not do on that specific cycle but rather did it towards the end of the development of the algorithm. I will discuss this further in the “I have a methodical approach to problem-solving” section of the review.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You have a good understanding of Ruby syntax and language constructs and are familiar with many of the built-in methods. You have a good grasp of the concept of array and string manipulation as well. The one thing that you could look at is rather using the map method instead of the each method when you need to manipulate an array that does not need to maintain the values from before being manipulated. The array for pushing the words to is a valid approach though.

I CAN DEBUG ANYTHING – Strong

Feedback: You were able to debug your code really quickly when running into problems as you used tools that are at your disposal such as reading the backtrace properly and interpreting where the error occurred and why it occurred very well. There was no point where you found yourself trying random changes to code in order to get it to work.

You used IRB quite well as well in order to check your assumptions before introducing code to the algorithm. This is a proactive way to reduce the introduction of bugs to the code.

I CAN MODEL ANYTHING – Strong

Feedback: You modelled your solution with a single method responsible for the spell-checking. A method without a class was sufficient as state is not really necessary to complete this exercise. By starting with a single method, you left your algorithm open to adding methods later when refactoring in order to keep your main method adhering to the single-responsibility principle.

Your method name “spell_check” adhered to the Ruby naming convention of snake_case and was also an actionable name that describes what the method does in a way that reflects the client’s domain. All of the other extracted methods also adhered to these conventions.

The algorithm that you completed made logical sense and was able to adhere to the requirements as were provided by the client and you were also able to start covering the punctuation requirements.

I CAN REFACTOR ANYTHING –Steady

Feedback: You were able to start refactoring your code towards the end of the development of the algorithm. Refactors that you had done were ternary operators as well as method extraction. A possible addition to your refactoring is to consider changing variable names from types such as “string” to “sentence” in order for the code to be more readable.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You have prioritised core cases over edge cases, thereby delivering immediate value to the client.

Your tests progressed in a logical order and at no point introduced too much complexity to the algorithm on any iteration of the RGR cycle.

You did not prioritise refactoring in your RGR cycle but opted to refactor at a later stage in order to get more test cases completed. I would encourage you to refactor when you identify it in order to avoid complex refactoring later which has an increased chance of introducing bugs.

You have also conducted some research during your development process just to make sure that you were using the correct method in Ruby for what you were trying to achieve. This is a great way to ensure that development resources are used efficiently and not used on code that you assume would work.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You did a great job gathering information about how the spell checker should work clearing up all of the finer details. You also made excellent use of an input-output table with your own examples in order to flesh out the behaviours a bit starting from the simplest cases to the more complex cases. This served you very well when you started creating your tests as you just copy-pasted these inputs and outputs into your tests thereby encoding the requirements as confirmed with the client into the tests. This ensured that your algorithm would adhere to all of the acceptance criteria.

I WRITE CODE THAT IS EASY TO CHANGE – Strong

Feedback: Your code is once again properly decoupled from your tests by testing for behaviours instead of implementation details. This is a great habit to maintain as I have discussed in the previous review.

Your method names are really descriptive and generally, your variable names are descriptive as well. This contributed to making your code readable, which in turn makes code easier to change. As I have mentioned in the “I can refactor anything” section of the review, try to use more descriptive parameter names than just the data type, as this can read to code that is hard to change because it is not as intuitive when reading.

You have now included Git commits on green and refactor phases which allows you to take advantage of the benefits that I discussed in the previous review. I would just suggest a small change which would be to add a capitalisation on the first word because that is in line with conventions.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You are generally really vocal and kept me updated with your progress and your decisions. I would, however, encourage you to justify any reasons for deviating from the process such as skipping a refactor and ensure that your reasoning is sound.