REVIEW FEEDBACK

Iain Aitken 30/04

30 April 2021 / 08:00 AM / Reviewer: Pierre Roodman

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

GENERAL FEEDBACK

Feedback: There is a lot of improvement to your process in this review session. You have taken the feedback from previous sessions and applied it to your process quite well. I would just encourage you to practice with similar problems in order to get some experience with what built-in Ruby methods to use for certain types of problems.

I CAN TDD ANYTHING – Strong

Feedback: You have once again used the information that you have gathered about the behaviours of the program in order to take a behaviour first approach to testing. This is good practice to maintain.

Your test progression is spot on and each new test that you introduced brought an incremental transformation to the algorithm.

You have adhered closely to the RGR cycle and were refactoring on every refactor phase in order to develop clean code on every iteration of the RGR cycle.

I CAN PROGRAM FLUENTLY – Steady

Feedback: You are very familiar with Ruby and RSpec syntax and are able to set up your environment and run tests with the terminal quite easily.

You seemed to have some difficulty with how to extract the operators and numbers. I suggest that you practice a bit with string and array methods by doing some exercises from a website such as CodeWars. The more you practice, the more you will build up an intuition for what a good approach to the problem would be.

After a bit of nudging in the right direction by suggesting using the split method, you did start making some progress in completing the exercise though.

I CAN DEBUG ANYTHING – Steady

Feedback: You are quite familiar with common errors and are able to determine if your tests are failing for the right reason or because of bugs in the code. You read the error messages quite well and are able to gather the important information from the backtrace. I would suggest that when you are having some problems using a built-in method, that instead of making some random changes to the code, that you perhaps check the correct usage from documentation or use IRB in order to test your assumptions such as with the to_f method. You were, however, able to figure out the problem through some trial and error though.

I CAN MODEL ANYTHING – Steady

Feedback: You have used a class with a single method which was a simple enough place to start and allowed for the extraction of methods later on which you have done in this review session.

You have followed Ruby conventions for method and variable names by using snake_case and including a verb in your method in order to make it actionable.

You had a bit of trouble getting your algorithm moving in the right direction, but after some nudging when you were a bit stuck, you were able to progress.

I CAN REFACTOR ANYTHING –Steady

Feedback: Your code was well refactored, although there were occasions when you created a method and started building logic for that method rather than creating the logic and then looking at whether you need to extract it into a new method. The approach that you are taking can lead to over-engineering, as you are making assumptions as to what methods you will need.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Steady

Feedback: You have prioritised core cases over edge cases in this review session. This is a good way to provide immediate value to the client.

Your tests were progressing in a logical order and each test brought a new transformation to the algorithm. With each new specific test, the code became more generic. I would just warn against doing three tests at once as you did towards the end, but your explanation of why you did it during my feedback at the end of the session made logical sense.

You have now also included refactoring your code on each iteration of the RGR cycle.

I USE AN AGILE DEVELOPMENT PROCESS – Strong

Feedback: You have conducted an outstanding, comprehensive information gathering session. You clarified all of the main requirements of the program and also did a great job discussing the edge cases. One thing that you could perhaps have asked about is how to handle zero division errors.

I would just suggest that you flesh out your input-output table a bit more with your own examples that you can use for what possible test progression may look like. You could then also make sure that the expected outputs for those cases are what the client is expecting.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: You made use of Git as part of your process, committing after your tests pass. This ensures that you have a working history of your code and that any rollbacks will return the working code. This also allows you the opportunity to keep a record of changes to your code, which helps a client to follow the progress of the program by reading the commit history. I would just suggest that you ensure that your commit messages are always specific as to what feature was completed with the passing of a test. An example is your first commit which speaks about a test for a basic sum passing could rather be something more specific to the actual requirements such as “Add feature for handling a single number as a sum with no operator”.

You once again had your test suite properly decoupled from the implementation. This was done by testing the expected inputs and outputs of the system. This promotes flexibility in your code, and so any changes made to the structure of the code will not warrant a change in tests.

Your naming for variables and methods were quite descriptive and made your code relatively easy to read and understand. I would just suggest the parameter of your calculate method rather being called “sum_input” as opposed to “input”.

I CAN JUSTIFY THE WAY I WORK – Steady

Feedback: You have communicated very well what you were doing and why you were doing it and you explained what phases you were on such as mentioning that you are looking at what you can refactor when on the refactor phases. I would just suggest that you justify any deviations from the process such as when you introduced three tests at once for the remaining three operators.