REVIEW FEEDBACK

Ben Gittins 25/03

25 March 2020 / 11:00 AM / Reviewer:

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

GENERAL FEEDBACK

Feedback: This task had quite a bit more complexity, but you employed good TDD and developed a minimum viable product. There's definitely some room for improvement in terms of information gathering, as there were quite a cases you did not consider. Most of these were edge cases, although there was definitely one core case you didn't address. Otherwise, you were pretty steady across most of the course goals, well done. I enjoyed this review.

I CAN TDD ANYTHING – Steady

Feedback: I was happy with your TDD process overall. You sometimes wanted to veer off and implement things with higher complexity than necessary, but you always caught yourself in time and ended doing the simplest thing, even if it was a bit clunky (which is fine, that's what refactors are for). You addressed all the core cases that you had captured during requirements gathering in your program. I was happy with your test progression.

I CAN PROGRAM FLUENTLY – Steady

Feedback: I was pretty confident in your programming ability. It was good to see that you were familiar with inbuilt methods like split. You thought up some interesting ways of dealing with the operators in the string and although the approach you wanted didn't work, I liked the way you were thinking as this

would have made the code very expandable. The approach you opted for in the end was entirely functional though, even if not as neat as you would like.

I CAN DEBUG ANYTHING – Steady

Feedback:

I CAN MODEL ANYTHING – Steady

Feedback: 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 considered doing a refactor to remove the clunky if-else statements to try to find a neater way of converting the string operators to actual operators. I think that whilst noble, you could have left these as is or perhaps refactored the if-elses to a switch case statement. I was happy with the idea behind this refactor, even if you couldn’t get it to work. You were looking for a way to both neaten the code and make it more expandable.

I was happy when you made a refactor to clean up the conversion to integers.

I HAVE A METHODICAL APPROACH TO SOLVING PROBLEMS – Improving

Feedback: I was pleased that you followed the tests you had laid out in your IO table, that you prioritised core cases over edge cases and that your tests covered all the core cases. In terms of information gathering you will need to be a little more methodical in terms of thrashing out the edge cases and full extent of program operation.

I USE AN AGILE DEVELOPMENT PROCESS – Improving

Feedback: This task required a more thorough information gathering process than the previous ones, as there were many more considerations to consider due to the different types you were required to work with. You asked good questions to establish most of the core cases. You asked about input and the expected output to clarify program operation. You asked which arithmetic operators could be used and whether negative numbers were allowed. To improve on this, you would need to ask more questions about how the input might vary. This would have established for you that there could be multiple numbers and operands in a sum, and that decimal numbers were also allowed.

In terms of edge cases, you captured the case that there may be only a single number in a string (with no operand) and that strings without a space in them were invalid. To improve here for next time, it would be nice to ask the user if there is a particular error they would like for the invalid input cases. You also should consider cases such as when the completely wrong input is supplied to the program, such as “What’s up calculator?!” instead of a string sum. In real life, we need our programs to be immune to all sorts of funny inputs, so that they throw errors which can be handled, rather than the program just crashing on an edge case which is not accounted for. You also need to consider how the input may break the program in other ways, such as divide by zero errors.

I WRITE CODE THAT IS EASY TO CHANGE – Steady

Feedback: I was happy with your Git usage. You initialised Git early on, committed after every test went green and supplied useful commit messages, making it easy for future developers to come in and change a feature. Note that after your refactor you only did git add and did not actually commit, it would have been good to commit when the test went green and after the refactor so you could always roll back to the last working version of your code if necessary.

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 – Steady

Feedback: It was nice that you gave the client updates on what you were going to do. I mostly 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. The only thing I couldn’t follow while you were doing it was the attempt to insert ‘\’ into the string, but I understood this from your explantation at the end.