1. **Identified a problem or inefficiency in a process related to quality or development, and implemented a change to improve it**

   Situation:
   I joined Afterpay in the middle of an overhaul of their Tech team towards Agile and DevOps processes. As part of this QAs became integrated into Dev teams and our team's Engineering Manager asked us to analyse our role and responsibilities to see how we could best fit into the new structure and help hit our new Continuous Deployment targets.

   What I did:
   I researched industry best practises and reviewed our current development process through the lens of the 4 DORA metrics. I presented my findings to the team for discussion with a few key suggestions:

   i. Move quality efforts earlier in the process through kick off sessions, PR reviews and local testing of branches before merge.
   ii. Remove approval after testing on QA server as a gate to production deployments. Master merge means a change has passed initial QA checks and is automatically deployed to Sandbox/Pre-prod.
   iii. Exploratory and manual testing on the QA server made optional. Not removed entirely because manual tests are still required in certain circumstances, for example: early PVT on integrations, infrastructure issues (secrets, proxies), transaction reconciliation.
   iv. Regression testing covered by the automation suite in CI and maintained by the whole team.

   Outcomes:
   We implemented the changes and successfully increased production deployment frequency from once a Sprint (3 weeks) to multiple times a week, reduced lead time for code changes from 2+ days to < 1 day and maintained our high levels of quality with less than 5% change failure rate (1 out of 20 deploys).

2. **Recognised a pattern of bugs repeating and implemented a change to stop the pattern from recurring**

   Situation:
   Reviewing past issues that had gotten into production I noticed that our team had a blind spot with non-critical issues. If an issue was large it would trigger our alerts or be raised as an incident and dealt with immediately, however smaller issues were either slow to reach the team or unreported. For example, a customer might call up our Customer Service (CS) team and complain they couldn't add a new card to their account which would be a symptom of an issue. However, CS may advise the customer to remove their existing cards and re-add the new card which would be a workaround. In this case there is no "incident" because the complaint was resolved but if this CS rep noticed this as a pattern of calls where could they report this to be investigated further? In our current system if the CS rep was thorough they may note it down and raise it with their Team Lead, who may raise it with the CS Manager, who may raise it with our team's Product Manager, who may raise it with our Engineering Manager, who would then raise it with the Dev team for investigation… This was definitely too many "mays" for a reliable process.

What I did:
We realised that the feedback loop from our customers was far too slow and we could get valuable information from the internal users of our APIs. To improve this we did two things:

i. Set up a Slack channel for the CS team and other internal users to have a direct line of communication to the team for non-incident level queries or issues.
ii. Created a weekly support roster with the responsibility of monitoring and resolving queries in this channel.

Outcomes: Feedback loop of customer complaints to our team reduced from weeks to days. Discovered issues existing in production that would have gone undetected improving our customers' experience. Improved our team's reputation internally with other teams. Helped us develop a less siloed mentality giving us a better understanding of how our work impacts other teams and customers.

3. Attempted to convince someone to change the way they were working, even though they didn't want to

Situation:
When I joined my current team the relationship between QA and Dev was not great. A major tension with a number of the team came from a perceived lack of value the QA process provided (vs comprehensive unit and integration tests) and how much it slowed down releases. A specific point of contention was an automated RestAssured suite QA developed and ran against live components deployed to a test server. The Dev Lead was adamant this was a waste of time because it was duplication of integration tests they had developed whereas the QA Lead had spent significant effort to establish the suite and felt the Dev team was biased and dismissive of QA's technical skills.

What I did:
As a new hire with no past history with either, I felt I had an opportunity to improve the relationship. I reached out to the Dev and QA Lead and offered to objectively analyse both sides of this and come to a conclusion which the whole team would use going forward. I analysed the coverage of all our automated tests (from unit to integration, end-to-end) and determined that the integration test method the developers were using (spinning up the component in a Docker container and hitting the APIs) was functionally the same as a real deployment (excepting infrastructure issues which we covered with smoke tests) but that the RestAssured framework created by the QAs did have better test coverage.

Outcomes:
Using the data I convinced the QA Lead to allow removal of the RestAssured framework and convinced the Dev Lead to allow QAs access to the main repo to integrate extra coverage and help maintain the integration tests. This reduced our CI test runs by 1 hour (which adds up on every commit) and encouraged greater collaboration between Dev and QA.
By approaching this issue objectively we built a good platform for the QA and Dev relationship going forwards. We didn't fight to keep something just because we built it but we acknowledged that the Dev team's creation served a similar purpose more efficiently and chose what was best for the team as a whole.

4. **When you are testing a new feature, under what circumstances would you deviate from a test script while performing manual testing?**

   When manually testing I view test scripts as a path but on the way to the destination there are many side paths which may not be explicitly mentioned but are important to check. For example, a script might lay out the steps and verify whether a user can log in to a website. But when logging in I might notice the button is not disabled when clicked so I would go down a side path and see whether I can click it multiple times causing multiple login attempts. The main path is "can a user login?" but I would deviate from this whenever a side path appears to investigate related behaviours (which is often).

5. **If you joined a team of 10 developers as the only Quality Engineer and could implement and change any process(es) you'd like, how would you ensure that the team delivers high quality software?**

   If I joined a team of 10 developers as the only QE the only way the team delivers high quality software is if the whole team cares about the quality of the code. So in my opinion the most important thing is establishing a team culture of good testing. This is easier said than done but I would work to make developers feel that their tests are just as important as the code they are delivering through reviews and feedback on the design and coverage (both positive and negative! Not many people give credit for a nice test...) and if required stepping in to assist. In terms of processes, a good testing culture feeds into strong automated tests (unit, integration, end-to-end) which I would ensure are run on every commit in the pipeline.