SDET Coding Challenge

Table of Contents

1. Intro

LoanPro has built a new, state of the art calculator that's set to revolutionize the world of basic arithmetic operations. As part of the team that built it, your job is to build the right tools, tests, processes and procedures to guarantee correctness. Your software developer peers tell you that it appears to be working OK for all intended purposes; there's even some unit tests providing coverage, so their confidence level is high. As a Software Developer Engineer in Testing you think it's great there's unit test coverage, but would like to perform other types of testing as well before releasing to the public.

2. Goals

The goal is to use any means necessary to uncover any and all bugs.

3. Deliverables

1. A document with all findings, so that developers can replicate bugs in their own environment
2. For each bug, provide hints into what might be happening that causes the bug
3. If you use automation tools or custom code, provide the source code and instructions on how to run those

4. Instructions for the calculator

The only requirement is to have Docker installed in your computer. It should work OK in macOS (apple silicon and intel) and Linux ($x86_{64}$ and arm64). Windows is not supported.

First pull the image:

```
docker pull public.ecr.aws/l4q9w4c5/loanpro-calculator-cli:latest
```

After pulling the image, execute it with:

```
docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli add 8 5
```

Available operations are:

- `add`
- `subtract`

- `multiply`
- `divide`

5. Known bugs

The following is a list of known bugs and/or issues that **should not** be included in the report.

1. All commands take exactly two numbers. Attempting to use more or less operands will result in an error or incorrect results; this is expected behavior.
2. Dividing by zero throws an exception.
3. Results are guaranteed exact up to 16 digits. For example adding `1.000000000000001 + 1.000000000000001` should yield the expected `2.000000000000002`. Increasing one additional `0` results in `2.0`. Similar rounding errors due to the data type used are expected.
4. Operation resulting in infinity, negative infinity or “not a number” are not supported.

Author: César Olea

Created: 2024-04-25 Thu 15:26

Validate