# Sugar Labs Testing

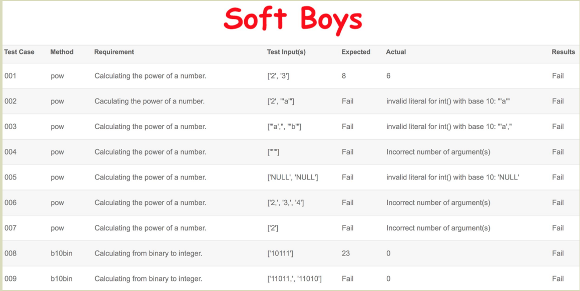
**Soft Boys**
Joe Ayers, Maz Little, Jasmine Mai, & Joe Spencer

## What is Sugar Labs?

- Learning platform
- Aimed at kids & young adults
- Interactive activities
- Self-Contained OS

## Testing Framework

Sugar Labs is essentially an operating system that is able to run different packages depending on the software. Therefore, we tested a package that works coincide with Sugar Labs which is mostly mathematical functions. The structure of our framework, as required, relies on folder structure. It is a shell method that runs with a set of inputs and is method- or test-case-agnostic. The script is able to take inputs from text files located in a test case folder. Below is a screenshot of the results.



Soft Boys

| Test Case | Method | Requirement | Test Input(s) | Expected | Actual | Results |
|---|---|---|---|---|---|---|
| 001 | pow | Calculating the power of a number. | ['2', '3'] | 8 | 6 | Fail |
| 002 | pow | Caculating the power of a number. | ['2', "'a'"] | Fail | invalid literal for int() with base 10: "'a'" | Fail |
| 003 | pow | Calculating the power of a number. | ["'a',", "'b'"] | Fail | invalid literal for int() with base 10: "'a'," | Fail |
| 004 | pow | Calculating the power of a number. | ['""'] | Fail | Incorrect number of argument(s) | Fail |
| 005 | pow | Calculating the power of a number. | ['NULL', 'NULL'] | Fail | invalid literal for int() with base 10: 'NULL' | Fail |
| 006 | pow | Calculating the power of a number. | ['2,', '3,', '4'] | Fail | Incorrect number of argument(s) | Fail |
| 007 | pow | Calculating the power of a number. | ['2'] | Fail | Incorrect number of argument(s) | Fail |
| 008 | b10bin | Calculating from binary to integer. | ['10111'] | 23 | 0 | Fail |
| 009 | b10bin | Calculating from binary to integer. | ['11011,', '11010'] | Fail | 0 | Fail |

## Fault Injection

It was relatively easy for us to assess the functions we selected for our test cases, and determine the best places to insert faults. We tried to approach fault insertion by making changes that could conceivably be the result of human error. These included functions such as factorize, binary conversion, and determining if a number is an integer. However, when we went into the code again to assess it for possible fault injection sites, we found that the logic still wasn't complicated enough for an injected fault to feel like it actually had an impact.