# CS 115 Lab 2: Testing and Debugging

As we have see in class, just writing code is usually only half the process. Very often, we are given code that claims to solve a given problem... except how do we know if that is correct? And if it isn't, how can we make it so?

In this lab, you will get practice working with tests and debugging, to expand your repertoire of tools for software development.

## Task 1: Validating Vali-dation

For the first task, we are considering the task of validating dates.

The `vali_date` function takes in two numbers, a month and a day, and returns whether or not these numbers form a valid date including on a leap year. For example, `vali_date(5, 15)` should return `True`, because 05-15 (May 15th) is a valid date.

We have already provided you with a number of implementations of `vali_date`. Not all of them are correct.

Your goal in this section is therefore to write tests to figure out which implementations are correct. This consists of two subparts.

First, implement test cases for `vali_date`. One such test case is already provided for you in `test_lab2.py`. Your task is to write more tests in addition to the provided test which test whether a `vali_date` implementation **completely and correctly implements the specifications below.**

- `vali_date(x, y)` takes in two integers. The code may assume that its inputs are integers, and does not need to check the input type.
- The first integer is a month number, the second is a day number.
- If that month and day form a valid date that can occur, return `True`.
- If that month and day do not form a valid date, return `False`.
- Dates in leap years are included, so `vali_date(2, 29)` should return `True`, but `vali_date(2,30)` should return `False` because there is no February 30th. For reference of which days are valid, you may consult timeanddate.com/calendar/?year=2028, as 2028 is our next leap year.

You are welcome to write as many tests as you like. We will be grading for correctness ("does each test expect behaviour that actually matches the specification") and for completeness ("do the tests as a whole cover all the possible scenarios the function may encounter").

Additionally, in the report, describe your testing strategy. How did you divide up the space of possible inputs? What were your partitions?

After you have written these tests, run them to verify the correctness of the provided functions. You can change which function you are testing by editing the final line of `lab2.py`. For example, if you would like to test `vali_date_impl2`, then edit the final line to `vali_date = vali_date_impl2`.

The provided functions can be found in `lab2.py` For each implementation, in your report, either note down that it worked correctly and passed all tests, or what tests it failed if it did not pass them. You don't have to identify why it failed the tests, just which ones it failed.

## CA Check-In

After you complete the above task, talk to the CA in your section. Show them the output of running your tests on `vali_date_impl2` and how you interpret the output.

## Task 2: Debugging

A function `vali_date_buggy_but_fixable` is provided in `lab2.py`. Your next task is to figure out what the error in that function is. Using whatever debugging process you choose to use, figure out what the error is.

In particular, you should **document your debugging process**. This should include the following:

1. How you detected and/or reproduced the error. If you have not observed any errors, go back to Task 1 and improve the coverage of your test cases.
2. How you came to understand how the function works.
3. How you isolated the exact point where the mistake was.
4. What that exact error was.

We will be primarily evaluating your process, especially how it would scale to debugging more complex code.

## Rubric

| Category | Description | Points |
|---|---|---|
| Name | For stating your name on every file for submission. | 2 |
| Pledge | For writing the full Stevens Honor Pledge on every file for submission. | 3 |
| Collaboration | For listing your collaborators. | 5 |
| Check-In | For performing the CA Check-In during the lab. | 10 |
| Task 1: Tests | For having a comprehensive testing suite, that can detect a variety of errors while passing correct code. | 30 |
| Task 1: Analysis | For filling out the report for Task 1. | 20 |
| Task 2: Debugging | For documenting a thorough debugging process which you used to locate the error. | 30 |
| **Total** | | 100 |

## Submission Instructions

For this lab, you must submit the following files, with exactly the following names:

- `test_lab2.py`
- `lab2_report.txt`

**Remember to put your name and the Stevens Honor Pledge on every file you submit!** Failure to put the pledge can result in marks being deducted, with repeat penalties potentially getting stricter between assignments.

Submissions must be handled through Gradescope (accessible through the homework page on Canvas). Grades will be released through Gradescope once the homework is graded.

## Notes and Reminders

- Your CAs are here to help you! Ask for help as soon as you need it.
- Think about how to categorize or group inputs when thinking of test cases.
- Consider possible edge cases. What are unusual inputs that could easily be missed in the code?
- Code that doesn't compile (throws a `SyntaxError` on being run) will receive a 0 for the assignment! Make sure you submit code that can be run, even if it does not include everything you want.
- **Don't forget to add your name and pledge!**