Lab 2: Assertions, Exceptions, and Tesing

2024-09-13

Table of contents

Today we will spend some time getting familiar with some of the programming tools that can help make your code more robust and resilient to errors and boundary conditions. Tools like unit tests, exceptions, and asserts (and next week we will spend some time on debugging misbehaving code).

Testing is what you do when you finish implementing a piece of code and want to try it out to see if it works. Running your code manually and seeing if it works is a workable strategy for simple one-time scripts that do simple tasks, but there are situations (like writing a function that others will repeatedly use, or like running the same piece of code on hundreds of files or URLs) where it is prudent to test your code ahead of its actual use or deployment. Today we will use the pytest package to do that. We will also use some error handling techniques to make sure our code can handle malformed inputs.

Lab Goals

In today's lab, we are going to get acquainted with: - raising exceptions; - setting up testing with pytest; - testing for correct behavior; - testing that exceptions are raised in the right situations.

Lab Exercise

1- Write a function in Python that finds a number in a string using regular expressions. The number can be positive or negative, integer-valued, or real, and numbers less than one can also be of the form 0.96 or .96. It can be assumed that no numbers appear in scientific notation. Refer to Using regex in Python for a quick overview of Python's `re` package.

2- Write an interface for that function (a function name and arguments), but do not implement the function yet (you can have it return an `None`, or an empty string for now). We will do this in a good old fashioned .py file (not a notebook or a quarto file).

3- Build a test suite using the `pytest` package to test that your function works as intended. Add at least 8 test cases with justification for each. Try to cover the main use cases, and as many potential corner cases or boundary conditions as possible.

4- Now run the test suite. It should fail for all your tests (unless one of them was passing an empty string).

5- Implement the function. You can do this at one go, or case by case. As you implement a case, you can rerun the test suite and see some of the tests relevant to that cases stopping to fail. When all the tests pass, you are done. This is called test-driven development.

6- If some cases are still failing, that's alright, we can use that failing code next week for demonstrating debugging functionality.

7- Make sure you raise an exception to trap invalid input types.

8- Add a test to check that an exception is properly raised when the function is given invalid input types.

Acknowledgements

This lab was originally authored by Ahmed Eldeeb and adapted for the Fall 2024 semester.