DS2000
Fall 2021
Handout: Functions + Testing

We already know the most important part of testing and debugging: when something goes wrong, DON'T PANIC!!

The second most important part of this process is to use our lovely ***modular, functional*** code to identify problems in small, isolated pieces. Gone are the days where we write test cases in comments, and then run an entire program end-to-end and see if it turns out OK! That's crazy now, because your programs are too sophisticated and complex, so we're going a better way.

Now that our code is modular, we can test the pieces of it and not just the whole thing:

- **A test suite** calls each of your functions a few times, and makes sure they return what we expect. It checks out each function by itself.
- **All functions are individually good?** Cool, now you can put them all together into one big overall program.

Structuring Your Code
We'll mostly have two files from now on: Your actual program, and a test suite.

In most cases, the test suite will be written by us, your friendly and helpful DS2000 teaching staff, and it will live on the Gradescope autograder. Sometimes you'll write your own test suite as well, and the point of the testing either way is to make sure each of your functions works on its own.

You'll write a Python file containing your actual program, with your functions defined above main as usual. The test suite will import your Python file and call your functions.

another for testing. We'll sometimes write the test_suite.py file, but you'll write your own as well. The test suite imports the other .py file.

A new main!
Does the main look a little different in the above image? Yep, and that's because we need to get all these files working together.

Our goal is to be able to do two things:

1. When we run the file *actual_program.py* the entire program runs from end-to-end. It probably does all the usual things we expect of our programs -- gathers data, does computations, maybe makes a plot. Just a normal program.
2. When we run the file *test_suite.py*, we want to be able to call the functions defined in *actual_program.py* but we **don't** want to run the entire thing end-to-end. We're just making sure the functions work individually, so we want to call them one at a time without other stuff getting in the way.

We do this with a new way of defining **main,** which will replace our **def main() / main()** structure from now on!

```
if __name__ == "__main__":
      # usual main stuff goes here
```

Super weird syntax, which takes some getting used to -- there are two underscores before and after the word *name* and the word *main*. But it does what we wanted, and allows us to run either of our two files without them getting in each other's way.

Here's how: Every time we run a .py file, it is given a name, which Python labels *__name__*.

- If the .py file we're running has been imported, its *__name__* is the module name.
- Otherwise, its *__name__* is *__main__*.

In the example above, there are two .py files we can run.

- If we run *test_suite.py*, then we've imported *actual_program.* All of the code and functions defined in *actual_program* are given their own name. All of the code and functions defined in *test_suite* are given the name *__main__*.
- If we run *actual_program.py*, then all the functions and code in actual_program have the name *__main__*. The other file, test_suite, just gets ignored.

Writing Tests

Not much you need to do here for now. When there are specific functions required as part of a homework, we'll write test suites into the autograder. As long as you've written your **main** in the new way and defined your functions as requested, you'll be all set.

What happens in these test suites? They look sort of like our tiny *test_suite.py* example above. In short, it's a Python program that:

- Imports the .py file where some functions have been defined.
- Calls each function 2-5 times with various arguments -- and in each case, we know what the answer should be.
- Prints out the result -- here's what we expected, here's what the function actually returned.
- At the end, reports how many tests passed/failed.

Whenever we write test suites for homework, we'll always share the source code so it's not just mysteriously hidden in the autograder. You can look and see what we're doing, and ask us questions if it's ever not clear.

For example, here is the test suite we wrote for HW5. It calls the three functions we'll ask you to define, each time with specific arguments, and figures out if the test passed or failed. You'll see that the code is pretty simple -- you can write your own tests, too, and will want to soon!