CIS 1100

Unit Testing

Python
Fall 2024
University of Pennsylvania

Reminders:

- Midterm 1 is coming up: this Wednesday October 9th in Class
  - No "notes sheet", we will provide an appendix for you of some things
  - Exam Review session TONIGHT 8:30pm - 10pm in Levine 101 (Wu & Chen Auditorium)
  - Midterm Clobber Policy: If you don't do as well as you want, you can get a grade improvement by doing better on the final exam.

Any questions?

Why Testing?

Testing is important to ensure the correctness of your code. Common industry practice is to write unit tests, you will see this if you keep programming outside of this course.

- Writing functions allows us to develop large programs in small pieces
  - functions should have clearly defined *purpose* and *intended behavior*
- Easier to formalize correctness for small pieces rather than large programs
  - hard to answer *"is my Caesar Cipher correct?"*
  - easier to answer *"does my string-to-symbol conversion work properly in this case?"*

Intro to Unit Testing

We already were writing "unit tests" of sort in the Caesar homework.

After you completed a function (e.g. `encrypt()`) we told you to make `main()` have:

```
print(encrypt("ET TU, BRUTE?", "G"))
```

and then we asked you to make sure that it printed

out `KZ ZA, HXAZK?` when the program was run

This didn't use python's built in unittest, but was made up of the same ingredients:

- **The Input(s)**: `"ET TU, BRUTE?"` and `"G"`
- **The Expected Output**: `KZ ZA, HXAZK?`
- **The Actual Output**: gotten from running the `encrypt` function
- **Compariing the Expected and Actual Output**: Having you look at the output and making sure it looks correct.

Example: `my_problems.py`

Consider I have the file `my_problems.py` which has the following function:

```python
# returns the most common letter in the string. Ignores case and return value is upper case.
# incase of a tie, returns one of the most common letters. None if string is empty
def most_common_letter(input_string):
```

Lets write an example test together!

`my_problems_tests.py`

Practice (L11)

Given the function we were working on, what are all the different kinds of input that may be worth testing?

Edge Cases

It is important to have a wide variety of tests to cover all cases.

- If we pass a test, we only know it works for those cases (those particular inputs)
- Cases that are not tested may not work
- We often want to test a variety of inputs and any "edge cases" (cases that are not common or obvious, but still should be handled correctly)

Edge Cases

Example: we have a function that takes a string and
we want to test that function. We should probably try:

- Strings of only letters
- Strings of odd and even length
- the empty string
- Strings with various sequences of characters and non-alphabetic characters
- etc.

Edge cases will vary based on what function we are testing. This is
just to try and explain what we mean with different "types" of input.

Practice (C12)

Given the following function header and comment, what are 3 different test cases that may be useful to test?

Can you write a python test for one of those cases?

```
# Returns the middle value of the list.
# If the length is even, it returns the second of the two middle values.
# if the length is empty, return None
def get_middle_value(input_list):
```

Practice (C14)

Consider the following function header:

```
# Given a string, returns a dictionary that contains all the words (split on whitespace)
# mapped onto how many times that word shows up in the string.
def get_word_counts(string):
```

Come up 3 different test cases, choose one of them and write the python test code for it

Practice (C16)

Now that we have written some tests, lets write the function itself.

Sometimes it actually helps to write the tests first, it can help with thinking about all the cases the code will have to handle.

```
# Given a string, returns a dictionary that contains all the words (split on whitespace)
# mapped onto how many times that word shows up in the string.
def get_word_counts(string):
```

Hint: `dict[key] = value` and `split()` may be useful

Why use `unittest` instead of just printing?

If we were able to get similar affects by just printing? Why use unittest?

- Helps keep the code organized: Which stuff is for testing and which stuff is part of the actual program
- Unlike what we did in Caesar, the test can check the output for us

Next time

- Lecture next time is the midterm exam!
  - Try to show up early / on time so we can make sure you get the full alloted time
- Lecture on Friday will be more on unit testing w/
  some Dictionary Practice that will be useful for HW04