UNIT TESTING

REMINDERS:

- Midterm 1 is coming up: Monday, February 23 in class
  - No "notes sheet", we will provide an appendix for you of some things
  - Exam Review session in class (and then after, until 3:45) on Friday
  - 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?

MAIN

So far in this class we have put all our code directly in the py file. Now that we know functions, we should follow better practices.

Starting with HW03 onwards, all code should go in a function.

- import statements should still be at the top of the py file outside of functions
- Comments can be outside of functions

HELLO WORLD W/ MAIN()

From now on we will have a function called `main()` that will be where we keep the code we previously did not put in a function

We also need to add an if statement to the bottom of our code. Do not forget it!

For example:

```
def main():
    print("Hello World!")

if __name__ == "__main__":
    main()
```

MAIN() PRACTICE (C12)

rewrite the following code so that we use the new style that uses:
`def main():` and `if __name__ == "__main__":`

```
import sys

# Prints all command line args
def print_args():
  for arg in sys.argv:
    print(arg)

print("Hello!")
print_args()
```

WHY MAIN()?

Writing our code in a `main()` function has a few benefits:

- Usually means our code is easier to read and better organized
- We can now import functions from the python files we write
  - this is the most important point
  - means we can import our own code into separate files for testing (or for the REPL)
- This is also just a style convention followed by most programming languages

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 are writing "unit tests" of a 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
- **Comparing 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:

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

Let's 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 (C'14)

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: list):
```

PRACTICE (C'16)

Consider the following function header:

```
# Given a string, returns a dictionary that contains all the words (spli
# 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

WHY USE UNITTEST INSTEAD OF JUST PRINTING?

If we were able to get similar effects 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