UNIT TESTING

REMINDERS:

- Midterm 1 is coming up: Wednesday, October 7 in class
  - Exam Review session in class (and then after, until 3:45) on Monday before
  - Remember to schedule accommodated exams with Weingarten ASAP. We can't do it for you!
  - 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.
- Personality Quiz Project due tonight at 11:59pm or tomorrow at 11:59pm with a late token.
  - Presentations Oct 5 & 6th. Sign up on Ed by today!

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 string parser correct?"*
  - easier to answer *"does my string-to-symbol conversion work properly in this case?"*

INTRO TO UNIT TESTING

You'll be writing unit tests as part of your homework from now on, but you've already been asked to test your work so far.

In Personality Quiz, we give you a quiz to run your program with and a set of answers to type in. We also show you what the correct output looks like. You can compare what happens when you run your program to the actual output to determine if your solution is correct or not.

D. Checkpoint

Try running `python personality_quiz.py lila_or_lenu.quiz` and give the answers `1`, `0`, `0`. This should mean that outcome `1`, or "Lenu", is the outcome that gets the highest number of points. Your final output should then look like this:

```
$ python personality_quiz.py lila_or_lenu.quiz
lila_or_lenu.quiz
Are You a Lila or a Lenu?
Lila Cerullo and Lenu Greco are the main characters of Elena Ferrante's Neapolitan Novels.
While they form a remarkable bond of friendship, they are very different people! Take
this quiz to determine whether you're more like Lila or more like Lenu.
```

In Personality Quiz, we give you a quiz to run your program with and a set of answers to type in. We also show you what the correct output looks like. You can compare what happens when you run your program to the actual output to determine if your solution is correct or not.

- **The Input(s)**: The Quiz file and the provided answers
- **The Expected Output**: "You're a Lenu..."
- **The Actual Output**: gotten from running your code on the right quiz with those answers
- **Comparing the Expected and Actual Output**: Having you look at the output and making sure it looks correct.

Testing is *thinking like a detective:* identifying differences between what should have supposedly happened and what actually happened.

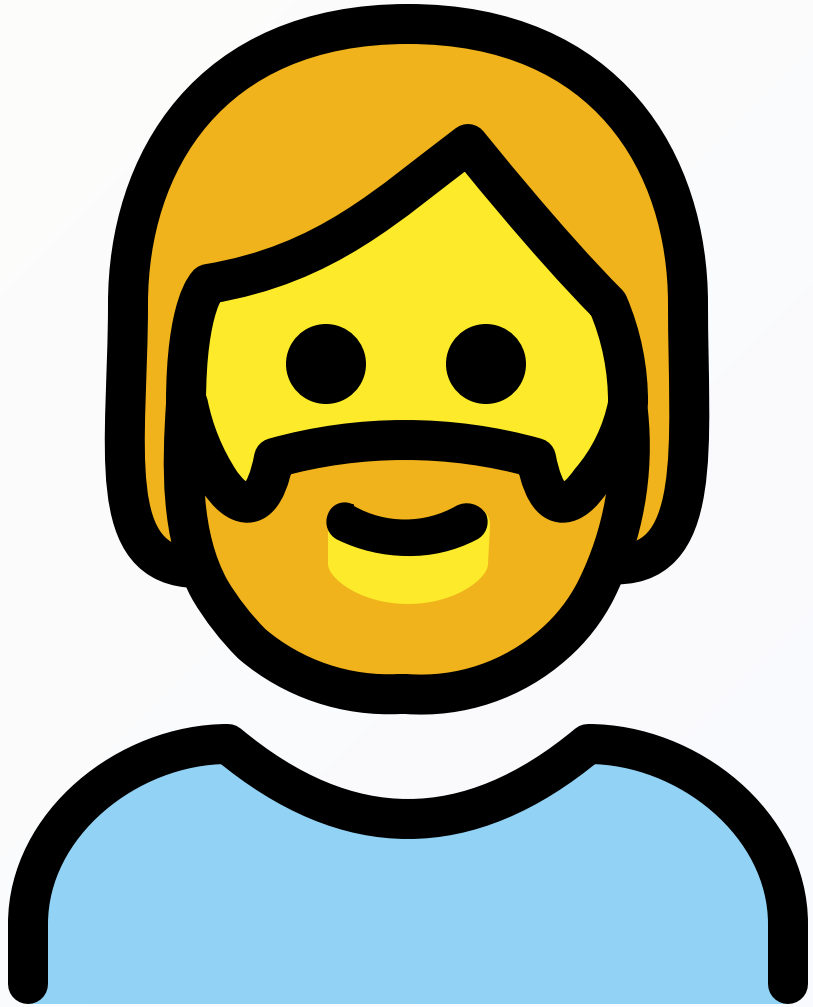
Let's Solve a Mystery!

A priceless Calder sculpture was stolen from the Philadelphia Museum of Art. At 8:22pm on Saturday, the power went out at the museum and the lights cut out. When the power was restored a minute later, the sculpture was gone!

ONE SUSPECT

Arthur, pictured at right, is known to be a particularly suspicious individual. (Just look at him!)

The PMA Detectives brought him in for questioning. They asked his whereabouts for Saturday evening.



ARTHUR’S TIMELINE

- "I went to see the new Resident Evil movie with my friend Natalia at the Ritz from 6-8pm."
- "Then, me and Natalia went to meet Kristian at Louie Louie at 8. We were there for a couple hours, 'til 10."
- "I got back to my apartment at the Bellevue at 10:15 and went to sleep then."

OUR EVIDENCE

- Natalia showed us receipts for two tickets to Resident Evil at the Ritz for the 6pm showing and claims that Kristian was there.
- Kristian posted an Instagram story with Arthur and Natalia outside Louie Louie on Saturday night.
- The sculpture went missing at 8:22pm from the Philadelphia Museum of Art
- The Bellevue's CCTV confirms that Arthur returned home at 10:04pm.

CATCH THE CULPRIT IN A LIE

You've got a bad feeling about Arthur

1. Do some legwork: work out more details about important elements from our evidence to get a fuller picture of what really happened on Saturday.
2. *Test* Arthur's alibi to catch him in a lie. Come up with a question to ask that reveals an inconsistency between his alibi and our ground truth.
    1. First, come up with some statement that we know to be consistent with our evidence.
    2. He will always answer consistent with his alibi, but he might say something inconsistent with our evidence.

- "I went to see the new *Resident Evil movie* with my friend Natalia at the *Ritz* from 6-8pm."
- "Then, me and Natalia went to meet Kristian at *Louie Louie* at 8. We were there for a couple hours, 'til 10."
- "I got back to my apartment at the *Bellevue* at 10:15 and went to sleep then."

- Natalia showed us receipts for two tickets to *Resident Evil* at the *Ritz* for the 6pm showing and claims that Kristian was there.
- Kristian posted an *Instagram story* with Arthur and Natalia outside *Louie Louie* on Saturday night.
- The sculpture went missing at 8:22pm from the *Philadelphia Museum of Art*
- The *Bellevue's* CCTV confirms that Arthur returned home at 10:04pm.

🎉🎉🎉 Congrats, you did it! 🎉🎉🎉

- **"Function" to test:** the alibi is like a function that returns answers based on questions asked.
- **inputs:** a question to the suspect
- **expected output:** some statement consistent with our evidence that required some detective work to generate
- **actual output:** the suspect's statement provided as an answer to the question that we asked; it will always be consistent with the alibi that they provided.

When the **expected** and **actual** outputs don't match up, we find a *fault* in the suspect's alibi.

A DETECTIVE'S MINDSET

We write unit tests to identify **faults** in our code.

- You know what the function should do (the **expected output**)
- You have some code that you wrote and you can't be sure what it actually does without running it (getting the **actual output**)
- You find a fault when the **actual** and the **expected** don't match up.

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 L1

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 C1

Given the following function header and comment, what are 3 different test cases that may be useful to test?
For each one, write an **input** and the output you would **expect** for a correct implementation.

```
# 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):
```

WORDLE

REQUIREMENTS:

We want a function called `score_guess(guess: str, hidden: str) -> str`.
It should represent a Wordle scorer and returns a string with:

- `"x"` for characters in the `guess` that are not part of the `hidden` word
- `"."` for characters in the `guess` that are in the `hidden` word but not in the right place
- `"+"` for characters in the `guess` that are in the right place in the `hidden` word.

ACCEPTING OR REJECTING IMPLEMENTATIONS C2

I need your help to build a **test suite**, or a comprehensive series of test cases that will help me identify whether or not an implementation of `score_guess` has any known faults.

Work with a partner and come up with as many **input/expected output** pairs as you can that cover different important cases.

- Don't forget edge cases!
- An **input** here is both a value for `guess` and a `hidden`.
- For example, one test case might be that `score_guess("happy", "happy")` should return `"+++++"`

AN IMPLEMENTATION

```
def score_guess(guess, hidden):
    result = ""
    for i in range(len(guess)):
        if guess[i] == hidden[i]:
            result += "+"
        if guess[i] in hidden:
            result += "."
        else:
            result += "x"
    return result
```

AN IMPLEMENTATION

```
def score_guess(guess, hidden):
    result = ""
    for i in range(len(guess)):
        if guess[i] in hidden:
            result += "."
        elif guess[i] == hidden[i]:
            result += "+"
        else:
            result += "x"
    return result
```

AN IMPLEMENTATION

```
def score_guess(guess, hidden):
    result = ""
    for i in range(len(guess)):
        if guess[i] == hidden[i]:
            result += "+"
        elif guess[i] in hidden:
            result += "."
        else:
            result += "x"
    return result
```

AN IMPLEMENTATION

```
def score_guess(guess, hidden):
    result = ["x"] * len(guess)
    remaining = []
    for i in range(len(guess)):
        if guess[i] == hidden[i]:
            result[i] = "+"
        else:
            remaining.append(hidden[i])
    for i in range(len(guess)):
        if result[i] != "+" and guess[i] in remaining:
            result[i] = "."
            remaining.remove(guess[i])
    return "".join(result)
```