Beat the Catme Training

Stanley So

sos@purdue.edu

LC4-12

**Introduction:**

**Why did I do this project?**

So one day I was just chilling and minding my own business until I saw that I had to do a Catme survey. I opened it and it said “training required”. The training took about 20 minutes but at the end of it I felt like it was quite meaningless and just a waste of my time.

I was chilling for another few weeks until I saw that I had to do the Catme training again for another class. I had enough of this training and was tired of my time being wasted, so instead of spending 20 minutes doing another Catme training I spent about 15 hours writing a program to give me the correct answers to every possible Catme survey.

**What does this project do?**

The title is pretty self explanatory, this program beats the catme training, meaning that it will give you the correct answers to any catme survey with 100% accuracy. (I don’t know for sure if it’s 100% accurate, but I’ve tested it thousands of times, and it has worked every single time.)

**Inputs and Outputs**

The user copy and pastes the 3 descriptions that are given to them on the second page of the Catme training. The program then outputs the answers to each question on the Catme training. The answers are printed in the form ((X, X, X), (X, X, X), (X, X, X), (X, X, X), (X, X, X)), where each question is separated by parentheses and the answers for each person are separated by commas.

**Description of All Functions**

Have fun reading through my 25 user-defined functions.

**Functions Inside calculate\_answers.py**

* calculate\_answers: Takes in a list of descriptions as the argument and returns the answers
* format\_description: Splits a description into a list of sentences and returns that list
* interpret\_data: Interprets the data in results.csv. Returns a dictionary with:
  + the keys being the sentences
  + the values being a list of tuples (if the sentence affects more than one question, we need to have more than one tuple)
  + the first index of the tuple being the question number
  + the second index of the tuple being what rating the sentence corresponds to
* calculate\_sentence\_rating: Given the sum of all the ratings and the amount of times that sentence appeared in the surveys, return the rating for that sentence.
* calculate\_paragraph\_ratings: Given one description, calculate the answer to each question

**Functions Inside get\_catme\_data.py**

* main: Gets a bunch of data from the Catme website for reverse engineering
* get\_results: Fills out one Catme survey and gets its data
* navigate\_to\_questions: Presses the "complete activity" button to get to the questions
* fill\_out\_question: Chooses an arbitrary answer for each person
* find\_reasons\_and\_rating: Finds out what the correct answer was for each person
* get\_reasons\_and\_rating: Returns a list of the reasons why that choice should have been the correct answer (Catme tells you what sentences should've affected your answer for each question)
* record\_reasons\_and\_rating: Saves the results into the results list (a global variable defined near the top of this program)
* go\_to\_next\_question: Presses the next button to go to the next question.
* find\_element: A version of [driver.find\_element](https://www.selenium.dev/documentation/webdriver/elements/finders/) that will signal the program to move on to the next test instead of throwing an error
* write\_results: Saves the results to results.csv

**Functions Inside input\_descriptions.py**

* main: Prompts the user for the 3 descriptions and then prints the correct answers

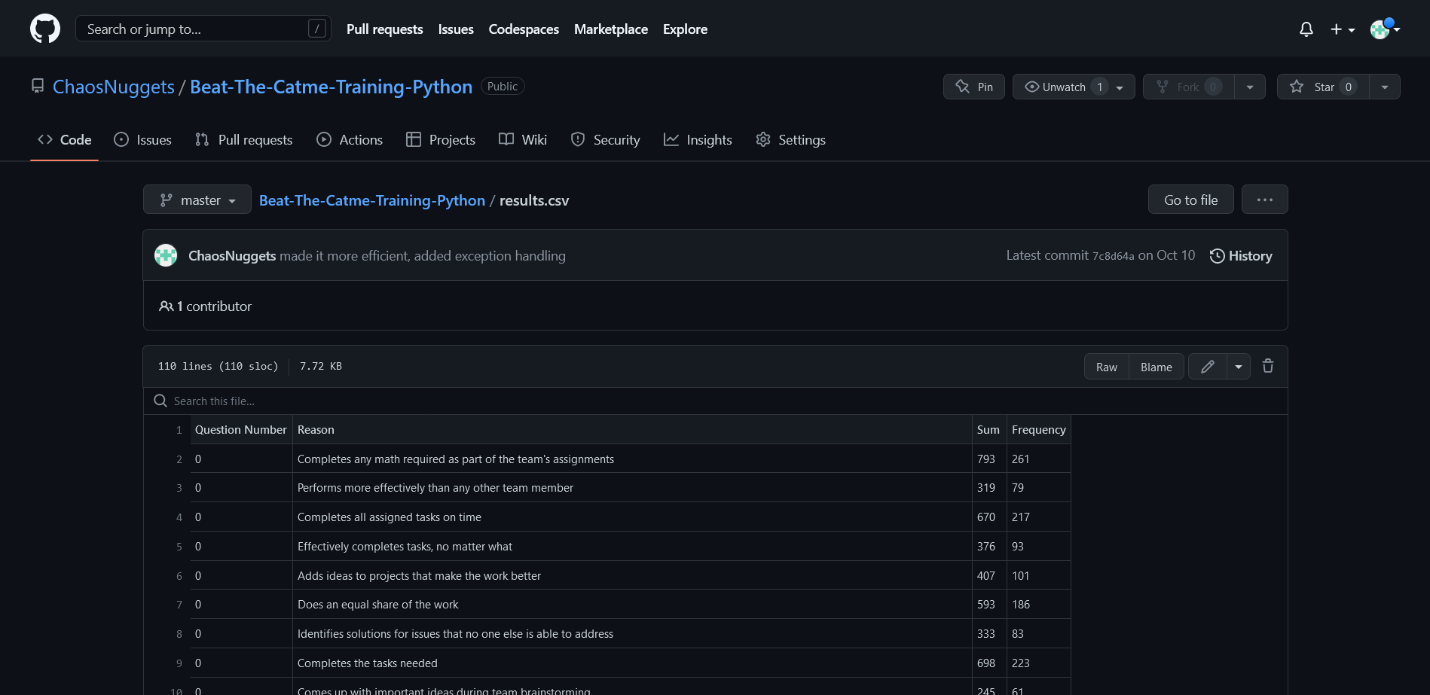
**Functions Inside test\_algorithm.py**

* main: Tests if calculate\_answers.py is actually correct by filling out TIMES\_TO\_RUN Catme surveys
* test\_algorithm: Fills out the correct answers for 1 Catme survey
* navigate\_to\_descriptions: Go to the first page of the Catme survey (which lists the descriptions)
* get\_descriptions: Return the descriptions for each person
* navigate\_to\_questions: Presses the "complete activity" button to get to the questions
* fill\_out\_questions: Given the correct answers as a parameter, it chooses the correct answer for each person
* go\_to\_next\_question: Clicks the next button to go to the next question in the Catme survey
* get\_score: Reads the text of the results page to see what score out of 30 we got
* find\_element: A version of [driver.find\_element](https://www.selenium.dev/documentation/webdriver/elements/finders/) that will signal the program to move on to the next test instead of throwing an error

**User Manual**

**Step 1: Getting the Catme Data**

The first thing you need for this program to work is the catme survey data. You can either download my data from [GitHub](https://github.com/ChaosNuggets/Beat-The-Catme-Training-Python/blob/master/results.csv), or you can get your own data directly from catme by running get\_catme\_data.py.

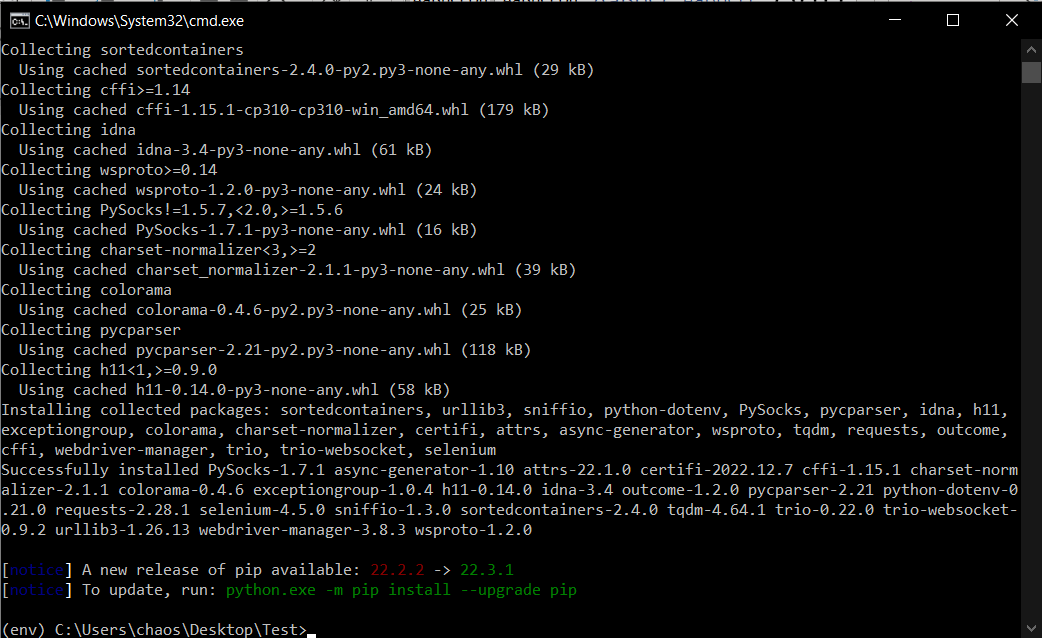
what results.csv looks like

**If You Decide to Get Your Own Data by Running get\_catme\_data.py**

In order to run get\_catme\_data.py, you need to have the selenium 4.5.0 and webdriver-manager 3.8.3 installed. To install these libraries, run these commands:  

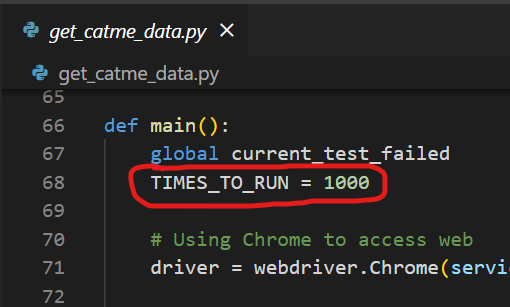

Alternatively you can download [requirements.txt from GitHub](https://github.com/ChaosNuggets/Beat-The-Catme-Training-Python/blob/master/requirements.txt) and run:



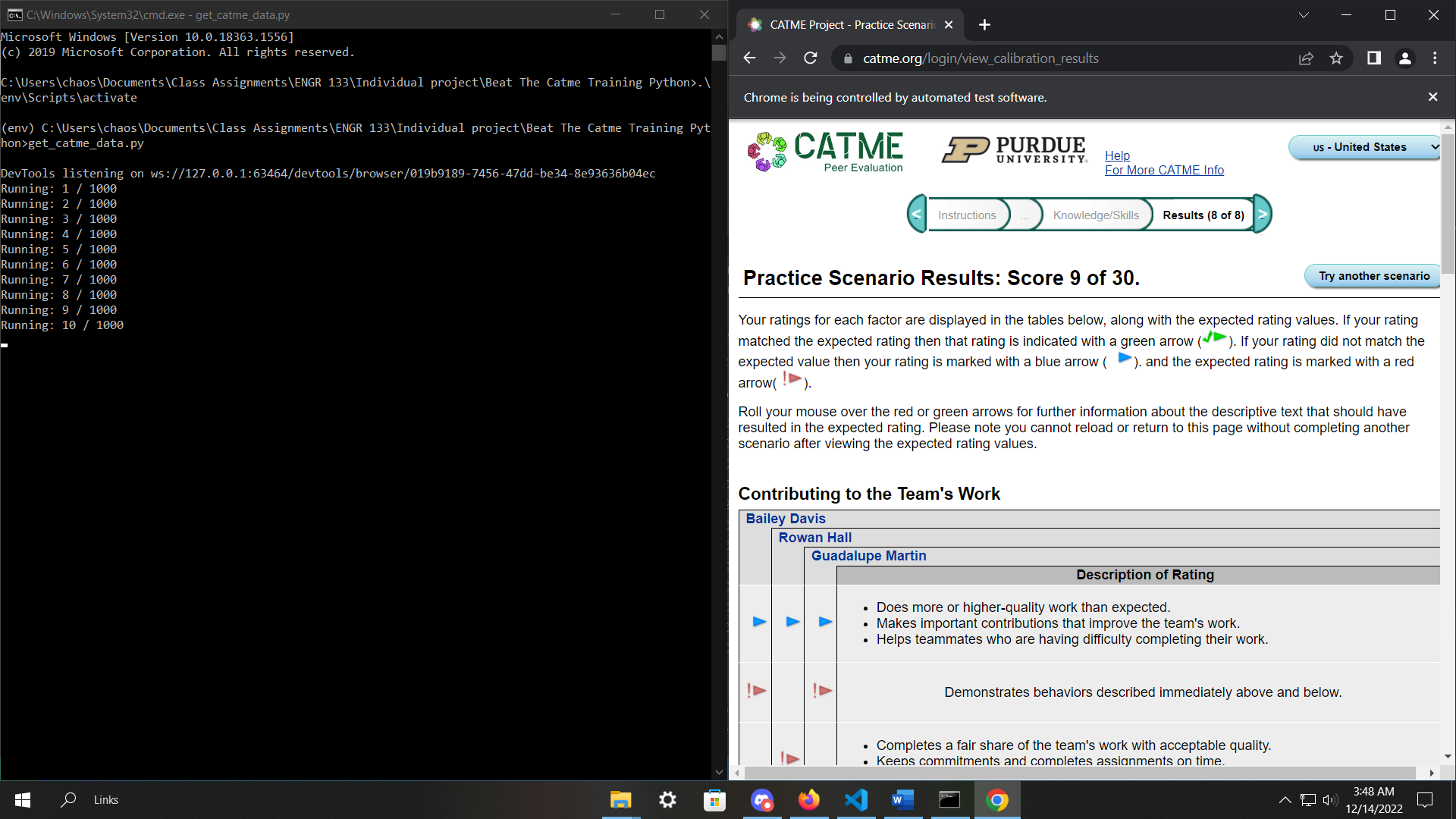


installing requirements with pip

Before running get\_catme\_data.py, you might want to tweak how many Catme surveys you want to download data from. You can do this by modifying the TIMES\_TO\_RUN variable on line 68. The default value is 1000. After this you can run get\_catme\_data.py normally.



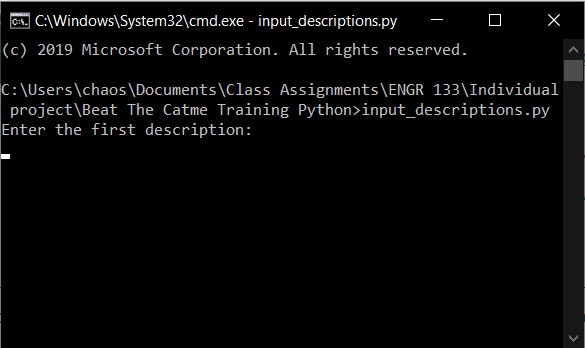
the TIMES\_TO\_RUN variable inside get\_catme\_data.py



what happens when you run get\_catme\_data.py

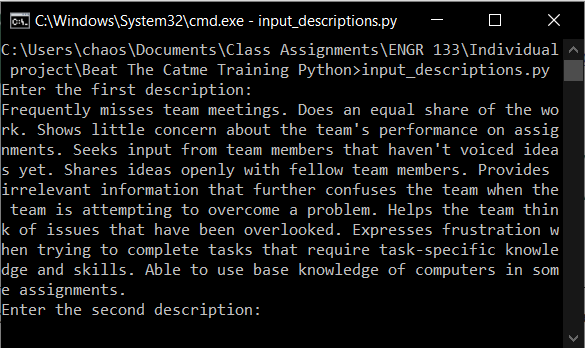
**Step 2: Getting the Answers**

To calculate the answers, run input\_descriptions.py. (Make sure that calculate\_answers.py and results.csv are in the same directory.) It will prompt you with text saying, “Enter the first description:”.



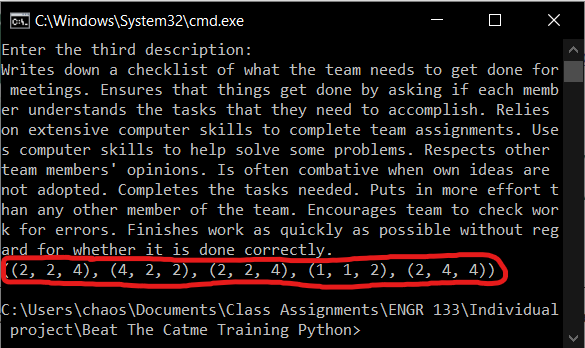
The first thing that will happen when you run input\_descriptions.py

Paste the first description and then press enter.



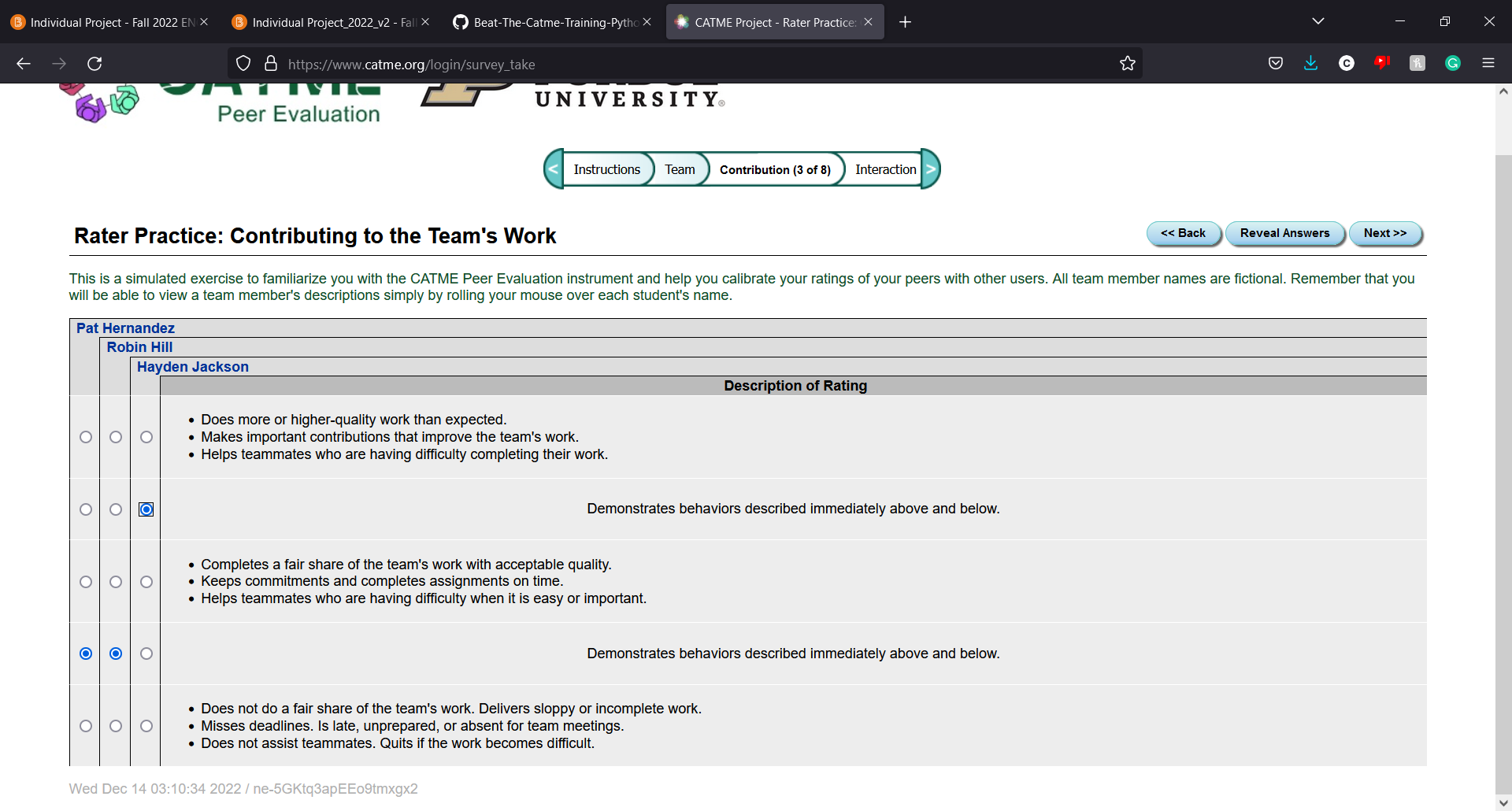
After you enter the first description

Enter the second and third descriptions. You will then see the answers printed in the form ((X, X, X), (X, X, X), (X, X, X), (X, X, X), (X, X, X)).



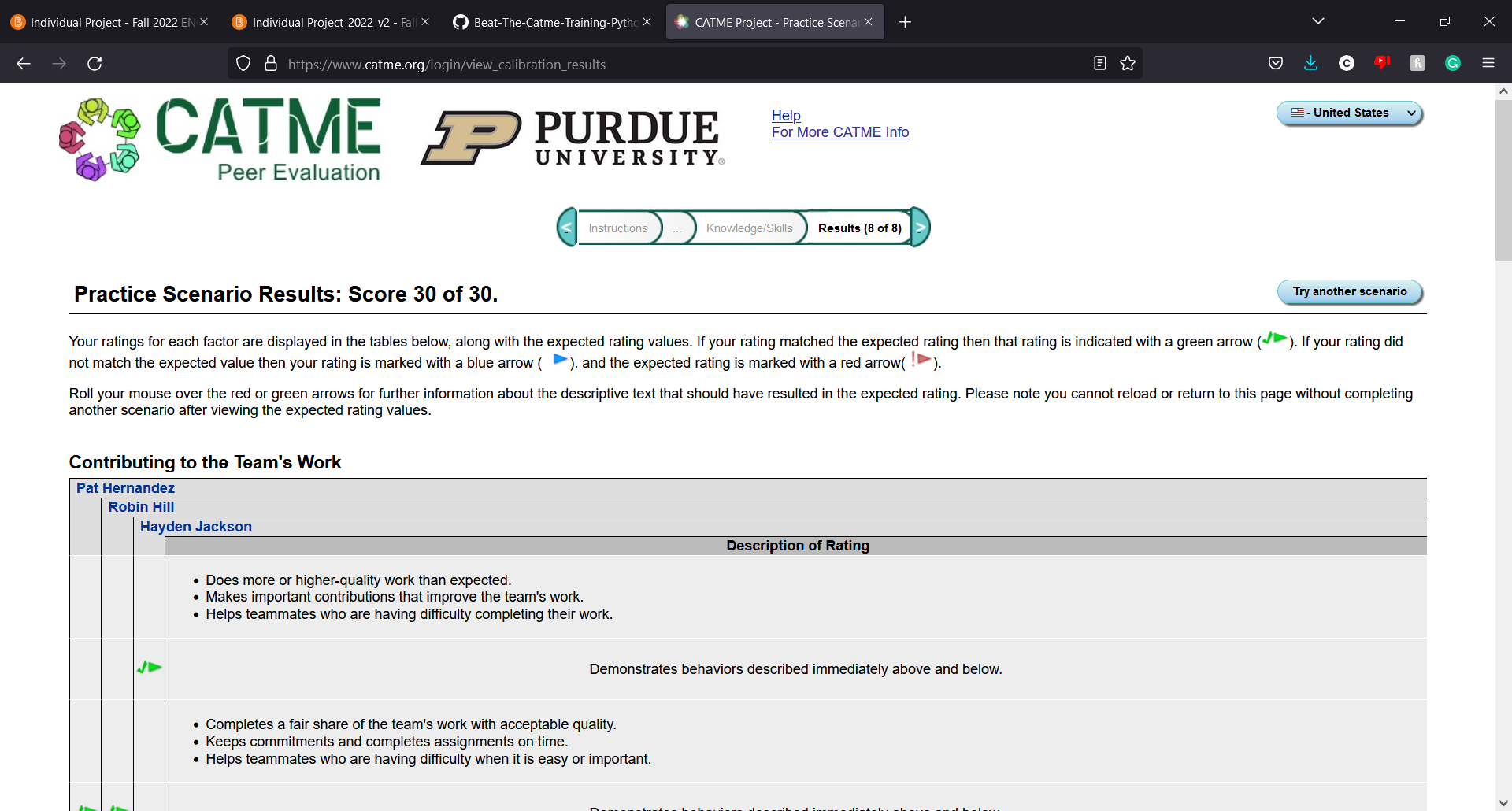
input\_descriptions.py answers

The answers are printed such that each question is separated by parentheses and each answer for each person is separated by commas. So based on these results, I should put the answers 2, 2, and 4 for question #1, 4, 2, and 2 for question #2, etc.



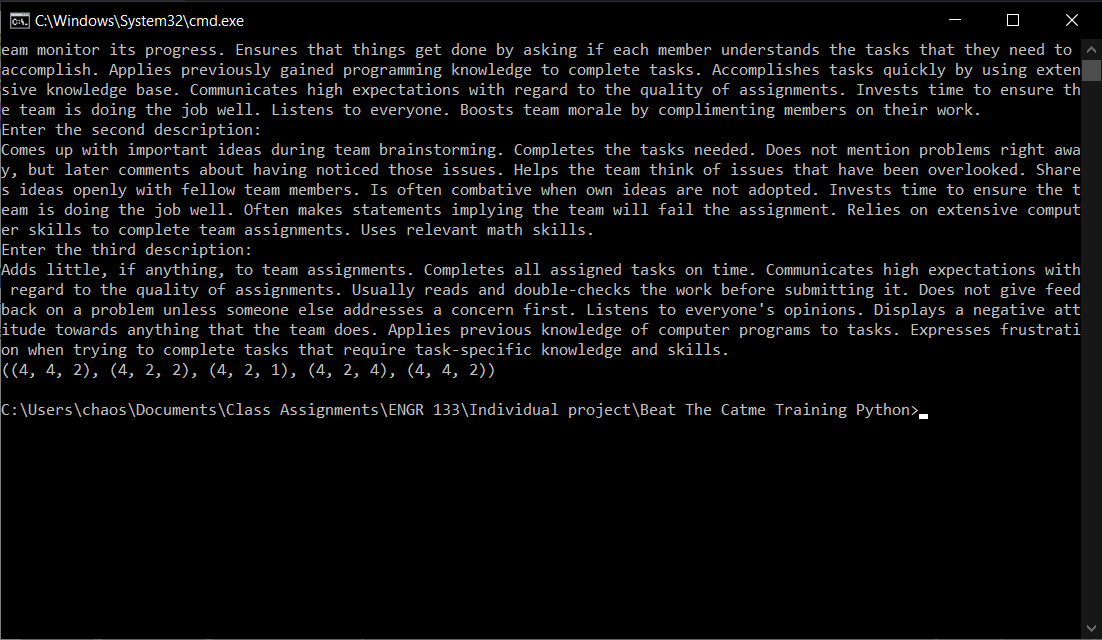
The correct answers for question #1

Repeat this for all 5 questions and you’ll get 30 out of 30 points.

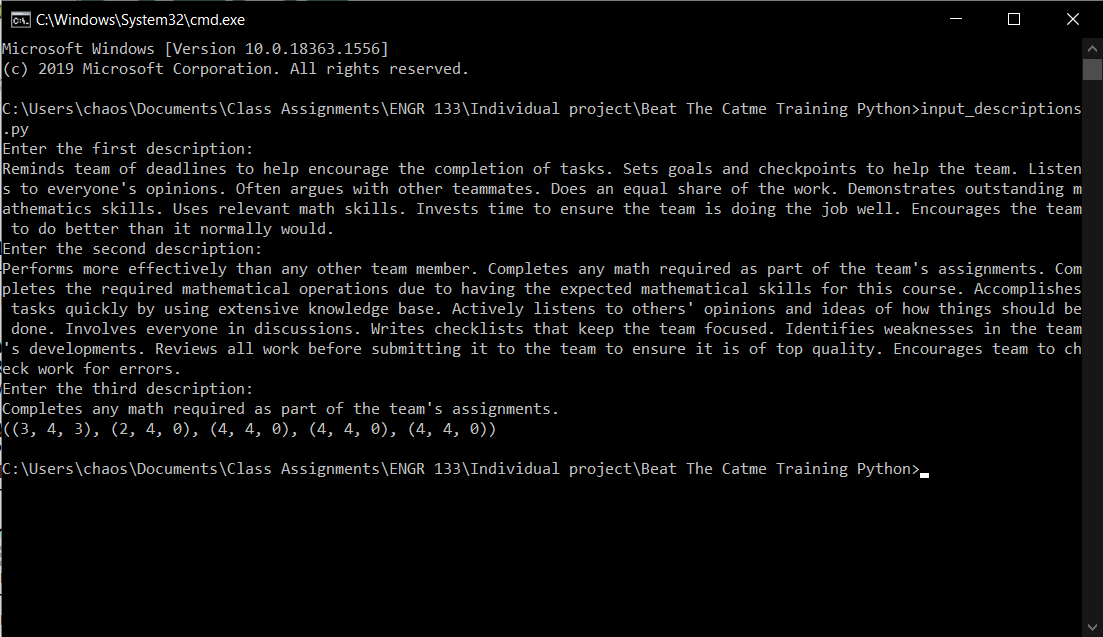


Catme training 30 out of 30 points

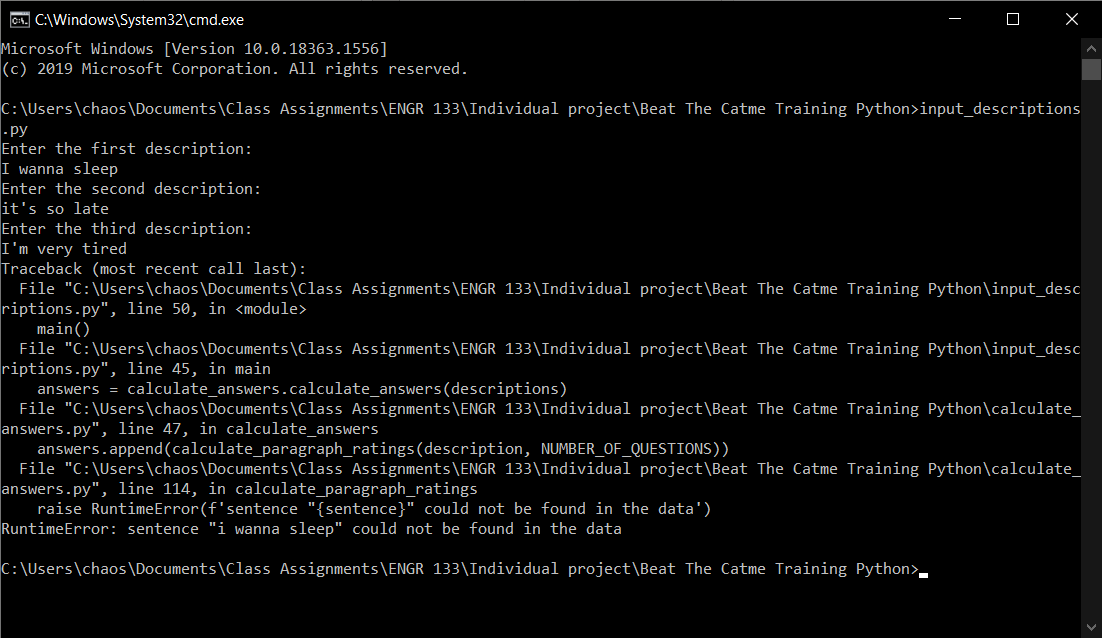
**Sample Inputs and Outputs**

****

a regular scenario where the user does everything correctly



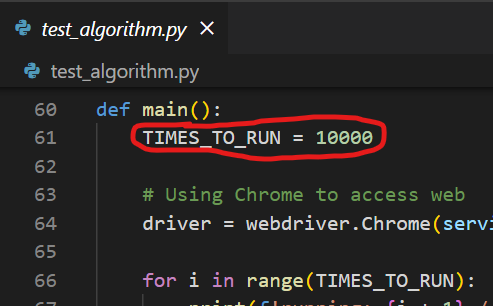
If the user enters and incomplete description, some of the questions don’t have enough information to be answered. This is shown by the 0’s.



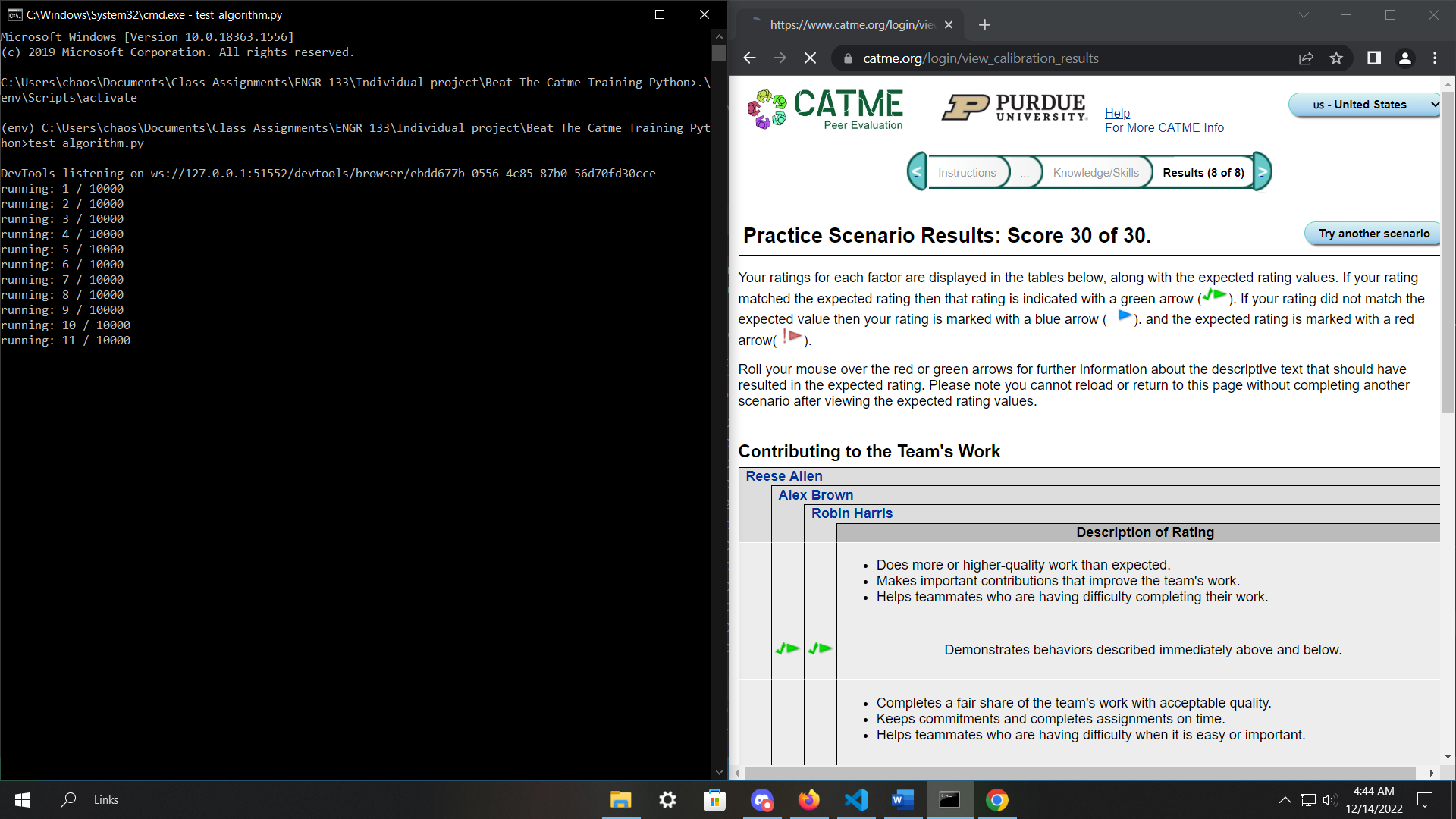
The sentence “I wanna sleep” is not a sentence that would appear in the catme training, so it’s not in our data. The program responds to this by throwing an error saying, “sentence ‘i wanna sleep’ could not be found in the data”.

**What if I wanted to test this program with hundreds or thousands of surveys?**

You can run test\_algorithm.py to automatically fill out Catme surveys. (Make sure that calculate\_answers.py and results.csv are in the same directory.) If you want to change the number of Catme surveys that are tested, you can change the value of TIMES\_TO\_RUN on line 61 of test\_algorithm.py.



TIMES\_TO\_RUN variable in test\_algorithm.py



what test\_algorithm.py looks like when ran

**What if I didn’t want to download anything but I still wanted to use this program?**

I rewrote the algorithm in JavaScript and put it on a website. <https://chaosnuggets.github.io/Beat-The-Catme-Training-Website/>

**Appendix**

**calculate\_answers.py**

"""

===============================================================================

ENGR 13300 Fall 2022

Program Description

    Has a bunch of functions that help calculate what the correct answers are

Assignment Information

    Assignment:     Individual project

    Author:         Stanley So, sos@purdue.edu

    Team ID:        LC4 - 12

Contributor:    Name, login@purdue [repeat for each]

    My contributor(s) helped me:

    [ ] understand the assignment expectations without

        telling me how they will approach it.

    [ ] understand different ways to think about a solution

        without helping me plan my solution.

    [ ] think through the meaning of a specific error or

        bug present in my code without looking at my code.

    Note that if you helped somebody else with their code, you

    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT

I have not used source code obtained from any other unauthorized

source, either modified or unmodified. Neither have I provided

access to my code to another. The project I am submitting

is my own original work.

===============================================================================

"""

# Import for type hinting

from typing import List, Dict, Tuple

# Import for reading csv file easier

import csv

# Takes in a list of descriptions as the argument and returns the answers

def calculate\_answers(descriptions: Tuple[int]) -> Tuple[Tuple[int]]:

    NUMBER\_OF\_QUESTIONS = 5

    answers = []

    # Calculate the answers for each of the descriptions

    for description in descriptions:

        description = format\_description(description)

        answers.append(calculate\_paragraph\_ratings(description, NUMBER\_OF\_QUESTIONS))

    # Make it so then the tuple that we return is tuple[n][m], where n is the question and m is the person

    return tuple(zip(\*answers[::]))

# Splits a description into a list of sentences and returns that list

def format\_description(description: str) -> List[str]:

    # Split the description into sentences

    description = description.strip('. ').lower().split('.')

    # Remove unnecessary whitespace

    for i in range(len(description)):

        description[i] = description[i].strip()

    return description

# Interprets the data in results.csv.

# Returns a dictionary with

# the keys being the sentences

# the values being a list of tuples (if the sentence affects more than one question, we need to have more than one tuple)

# the first index of the tuple being the question number

# the second index of the tuple being what rating the sentence corresponds to

def interpret\_data() -> Dict[str, List[Tuple[int, int]]]:

    data = {}

    with open('results.csv', 'r') as file:

        # Skip the first line

        file.readline()

        # Open the file with csv.reader to make it easier to separate data into columns

        csv\_reader = csv.reader(file)

        for line in csv\_reader:

            # Give all the data names

            question\_num = int(line[0])

            sentence = line[1].lower()

            summation = int(line[2])

            frequency = int(line[3])

            rating = calculate\_sentence\_rating(summation, frequency)

            # Add the data to the data dictionary

            if sentence in data:

                data[sentence].append((question\_num, rating))

            else:

                data[sentence] = [(question\_num, rating)]

    return data

# Given the sum of all the ratings and the amount of times that sentence appeared in the surveys,

# return the rating for that sentence.

def calculate\_sentence\_rating(summation: int, frequency: int) -> int:

    ratio = summation / frequency # In python we don't have to worry about integer division lol

    if ratio > 4:

        return 5

    if ratio == 4:

        return 4

    if ratio > 2:

        return 3

    if ratio == 2:

        return 2

    return 1

# Given one description, calculate the answer to each question

def calculate\_paragraph\_ratings(description: List[str], NUMBER\_OF\_QUESTIONS: int) -> List[int]:

    # The answer for each question from 1-5

    paragraph\_ratings = [0] \* NUMBER\_OF\_QUESTIONS

    for sentence in description:

        if sentence not in data:

            raise RuntimeError(f'sentence "{sentence}" could not be found in the data')

        for result in data[sentence]:

            # Give the data names

            (question\_num, rating) = result

            if paragraph\_ratings[question\_num] == 0: # if no other sentence has been found yet that affects the answer to that question

                paragraph\_ratings[question\_num] = rating

                continue

            rating\_sum = paragraph\_ratings[question\_num] + rating

            # Change the paragraph rating to the correct thing

            if 6 < rating\_sum < 10:

                paragraph\_ratings[question\_num] = 4

            elif 2 < rating\_sum < 6:

                paragraph\_ratings[question\_num] = 2

    return paragraph\_ratings

# Interpret and organize the data

data = interpret\_data()

# data\_str = str(data)

# print(data\_str.replace(')], ', ')],\n'))

**get\_catme\_data.py**

"""

===============================================================================

ENGR 13300 Fall 2022

Program Description

    Gets a bunch of data from the Catme website for reverse engineering

Assignment Information

    Assignment:     Individual project

    Author:         Stanley So, sos@purdue.edu

    Team ID:        LC4 - 12

Contributor:    Name, login@purdue [repeat for each]

    My contributor(s) helped me:

    [ ] understand the assignment expectations without

        telling me how they will approach it.

    [ ] understand different ways to think about a solution

        without helping me plan my solution.

    [ ] think through the meaning of a specific error or

        bug present in my code without looking at my code.

    Note that if you helped somebody else with their code, you

    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT

I have not used source code obtained from any other unauthorized

source, either modified or unmodified. Neither have I provided

access to my code to another. The project I am submitting

is my own original work.

===============================================================================

"""

# Import selenium for web driving

from selenium import webdriver

# Import By for finding by XPath

from selenium.webdriver.common.by import By

# Helps when using try catch when trying to find an element

from selenium.common.exceptions import NoSuchElementException

# Import the stuff to download the Chrome driver

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

# Import for type hinting

from typing import List, Tuple

# The number of questions the catme asks

NUMBER\_OF\_QUESTIONS = 5

# Seconds to wait before trying to locate an element again

TRY\_AGAIN\_TIME = 5

# The 1st dimension of the list are the different questions.

# The keys are the sentences that affect that question.

# The values[0] are the sum of all the ratings based on that sentence.

# The values[1] are the number of times that question has showed up.

results = [{}] \* NUMBER\_OF\_QUESTIONS

# The number of failed tests

failed\_tests = 0

# Whether or not the current test has failed

current\_test\_failed = False

def main():

    global current\_test\_failed

    TIMES\_TO\_RUN = 1000

    # Using Chrome to access web

    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    for i in range(TIMES\_TO\_RUN):

        current\_test\_failed = False

        print(f'Running: {i + 1} / {TIMES\_TO\_RUN}')

        get\_results(driver)

    write\_results()

    print(f'Done! ({failed\_tests} / {TIMES\_TO\_RUN} tests failed)')

# Fills out one Catme survey and gets its data

def get\_results(driver) -> None:

    global current\_test\_failed

    navigate\_to\_questions(driver)

    # Fill out each of the questions

    for i in range(NUMBER\_OF\_QUESTIONS):

        fill\_out\_question(driver)

        go\_to\_next\_question(driver)

    # Get the results

    for i in range(NUMBER\_OF\_QUESTIONS):

        find\_reasons\_and\_rating(driver, i)

# Presses the "complete activity" button to get to the questions

def navigate\_to\_questions(driver) -> None:

    global current\_test\_failed

    # Open the website

    driver.get('https://www.catme.org/login/survey\_demo\_team')

    # Find and click on list of courses

    complete\_activity\_button = find\_element(driver, 'name', 'action')

    if current\_test\_failed: return

    complete\_activity\_button.click()

# Chooses an arbitrary answer for each person

def fill\_out\_question(driver) -> None:

    global current\_test\_failed

    # Find and click a rating for each person

    person\_1\_button = find\_element(driver, 'name', 'person0')

    if current\_test\_failed: return

    person\_2\_button = find\_element(driver, 'name', 'person1')

    if current\_test\_failed: return

    person\_3\_button = find\_element(driver, 'name', 'person2')

    if current\_test\_failed: return

    person\_1\_button.click()

    person\_2\_button.click()

    person\_3\_button.click()

# Finds out what the correct answer was for each person

def find\_reasons\_and\_rating(driver, question: int) -> None:

    global current\_test\_failed, failed\_tests

    # The number of rows that we can choose

    NUMBER\_OF\_ROWS = 5

    # The people we still haven't found the correct answer for yet

    not\_found\_yet = [0, 1, 2]

    # Iterate through each row

    for i in range(NUMBER\_OF\_ROWS):

        # Create a temporary not\_found\_yet because removing elements from the actual not\_found\_yet

        # List while still in the for loop will cause weird stuff to happen

        temp\_not\_found\_yet = not\_found\_yet.copy()

        # Get the row

        row = find\_element(driver, By.XPATH, f'//section/div/table[{question + 1}]/tbody/tr[{i+5}]')

        if current\_test\_failed: return

        # Test if that row was the correct answer for any of them

        for j in not\_found\_yet:

            # Test if the correct answer is in that row

            try:

                reasons = row.find\_element('id', f'info{j}{question + 1}').get\_attribute('textContent')

            except NoSuchElementException:

                continue

            # Do these if the correct answer is in that row

            temp\_not\_found\_yet.remove(j)

            reasons\_list, rating = get\_reasons\_and\_rating(reasons, i)

            record\_reasons\_and\_rating(question, reasons\_list, rating)

        # Make the changes

        not\_found\_yet = temp\_not\_found\_yet.copy()

        # If we've found everything (the not\_found\_yet list is empty), then we can return

        if not not\_found\_yet:

            return

    print("couldn't find correct answer, moving on to next test")

    failed\_tests += 1

    current\_test\_failed = True

# Returns a list of the reasons why that choice should have been the correct answer

# (Catme tells you what sentences should've affected your answer for each question)

def get\_reasons\_and\_rating(reasons: str, row\_num: int) -> Tuple[List[str], int]:

    # Calculate the rating based on the current row

    rating = 5 - row\_num

    # Remove the unnecessary content from the text

    reasons = reasons.replace("The behaviors described in the phrase '", "")

    reasons = reasons.replace(".' should have resulted in the rating described for this factor.", "")

    # Split the text into its sentences

    reasons\_list = reasons.split('.')

    for i in range(len(reasons\_list)):

        # Remove unnecessary whitespace

        reasons\_list[i] = reasons\_list[i].strip()

    return reasons\_list, rating

# Saves the results into the results list (a global variable defined near the top of this program)

def record\_reasons\_and\_rating(question: int, reasons\_list: List[str], rating: int) -> None:

    # Copy the corresponding dictionary (I hate that I have to do this stupidity)

    question\_results = results[question].copy()

    # Iterate through each reason in reasons

    for reason in reasons\_list:

        # Add the reason to the results dictionary

        if reason in question\_results:

            question\_results[reason][0] += rating

            question\_results[reason][1] += 1

        else:

            question\_results[reason] = [rating, 1]

    # Add the changed dictionary back into results

    results[question] = question\_results

# Presses the next button to go to the next question.

def go\_to\_next\_question(driver) -> None:

    global current\_test\_failed

    # Find and click the next button

    next\_button = find\_element(driver, By.XPATH, '//form[2]/section/table/tbody/tr/td[3]/input')

    if current\_test\_failed: return

    next\_button.click()

# A version of driver.find\_element that will signal the program to move on to the

# next test instead of throwing an error (now that I think about it I could've just surrounded)

# lines 74 - 76 with try catch and it would've been so much easier)

def find\_element(driver, find\_method, method\_value: str):

    global current\_test\_failed, failed\_tests

    if current\_test\_failed: return

    try:

        return driver.find\_element(find\_method, method\_value)

    except NoSuchElementException:

        # Print the error and move on

        print('ran into NoSuchElementException, moving on to next test')

        failed\_tests += 1

        current\_test\_failed = True

# Saves the results to results.csv

def write\_results() -> None:

    with open('results.csv', 'w') as file:

        file.write('Question Number,Reason,Sum,Frequency\n')

        for i in range(len(results)):

            for key in results[i]:

                file.write(f'{i},')

                file.write(f'"{key}",')

                file.write(f'{results[i][key][0]},')

                file.write(f'{results[i][key][1]}\n')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**input\_descriptions.py**

"""

===============================================================================

ENGR 13300 Fall 2022

Program Description

    Prompts the user for the 3 descriptions and then prints the correct

    answers

Assignment Information

    Assignment:     Individual project

    Author:         Stanley So, sos@purdue.edu

    Team ID:        LC4 - 12

Contributor:    Name, login@purdue [repeat for each]

    My contributor(s) helped me:

    [ ] understand the assignment expectations without

        telling me how they will approach it.

    [ ] understand different ways to think about a solution

        without helping me plan my solution.

    [ ] think through the meaning of a specific error or

        bug present in my code without looking at my code.

    Note that if you helped somebody else with their code, you

    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT

I have not used source code obtained from any other unauthorized

source, either modified or unmodified. Neither have I provided

access to my code to another. The project I am submitting

is my own original work.

===============================================================================

"""

import calculate\_answers

def main():

    description\_names = ['first', 'second', 'third']

    descriptions = []

    # Prompt the user for the descriptions

    for name in description\_names:

        descriptions.append(input(f'Enter the {name} description:\n'))

    answers = calculate\_answers.calculate\_answers(descriptions)

    print(answers)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**test\_algorithm.py**

"""

===============================================================================

ENGR 13300 Fall 2022

Program Description

    Tests if calculate\_answers.py is actually correct by filling out

    TIMES\_TO\_RUN Catme surveys

Assignment Information

    Assignment:     Individual project

    Author:         Stanley So, sos@purdue.edu

    Team ID:        LC4 - 12

Contributor:    Name, login@purdue [repeat for each]

    My contributor(s) helped me:

    [ ] understand the assignment expectations without

        telling me how they will approach it.

    [ ] understand different ways to think about a solution

        without helping me plan my solution.

    [ ] think through the meaning of a specific error or

        bug present in my code without looking at my code.

    Note that if you helped somebody else with their code, you

    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT

I have not used source code obtained from any other unauthorized

source, either modified or unmodified. Neither have I provided

access to my code to another. The project I am submitting

is my own original work.

===============================================================================

"""

# Import selenium for web driving

from selenium import webdriver

# Import the stuff to download the Chrome driver

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

# Import By for finding by XPath

from selenium.webdriver.common.by import By

# Helps when using try catch when trying to find an element

from selenium.common.exceptions import NoSuchElementException

# Import for pausing

import time

# Import for type hinting

from typing import List

# Import our algorithm for calculating the correct answers

import calculate\_answers

current\_test\_failed = False

failed\_tests = 0

NUMBER\_OF\_QUESTIONS = 5

def main():

    TIMES\_TO\_RUN = 10000

    # Using Chrome to access web

    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    for i in range(TIMES\_TO\_RUN):

        print(f'running: {i + 1} / {TIMES\_TO\_RUN}')

        test\_algorithm(driver)

# Fills out the correct answers for 1 Catme survey

def test\_algorithm(driver) -> None:

    global current\_test\_failed

    current\_test\_failed = False

    navigate\_to\_descriptions(driver)

    descriptions = get\_descriptions(driver)

    answers = calculate\_answers.calculate\_answers(descriptions)

    navigate\_to\_questions(driver)

    # Fill out each of the questions

    for i in range(NUMBER\_OF\_QUESTIONS):

        fill\_out\_questions(driver, answers[i])

        go\_to\_next\_question(driver)

    try:

        if get\_score(driver) < 30:

            print('we made a mistake somewhere')

            time.sleep(9999999)

    except ValueError:

        # Move to next test

        print('ran into ValueError, moving on to next test')

        failed\_tests += 1

        current\_test\_failed = True

# Go to the first page of the Catme survey (which lists the descriptions)

def navigate\_to\_descriptions(driver) -> None:

    # Open the website

    driver.get('https://www.catme.org/login/survey\_demo\_team')

# Return the descriptions for each person

def get\_descriptions(driver) -> List[str]:

    NUMBER\_OF\_DESCRIPTIONS = 3

    # Get the descriptions and fill the list

    descriptions = []

    for i in range(NUMBER\_OF\_DESCRIPTIONS):

        description = find\_element(driver, By.XPATH, f'//section/dl/dd[{i + 1}]').get\_attribute('textContent')

        if current\_test\_failed: return

        descriptions.append(description)

    return descriptions

# Presses the "complete activity" button to get to the questions

def navigate\_to\_questions(driver) -> None:

    global current\_test\_failed

    # Find and click on list of courses

    complete\_activity\_button = find\_element(driver, 'name', 'action')

    if current\_test\_failed: return

    complete\_activity\_button.click()

# Given the correct answers as a parameter, it chooses the correct answer for each person

def fill\_out\_questions(driver, answers: List[int]) -> None:

    NUMBER\_OF\_ROWS = 5

    # Find and click a rating for each person

    for i in range(len(answers)):

        person\_i\_button = find\_element(driver, By.XPATH, f'//form[2]/section/div/table/tbody/tr[{(NUMBER\_OF\_ROWS - answers[i]) + 5}]/td[{i + 1}]/input')

        if current\_test\_failed: return

        person\_i\_button.click()

# Clicks the next button to go to the next question in the Catme survey

def go\_to\_next\_question(driver) -> None:

    # Find and click the next button

    next\_button = find\_element(driver, By.XPATH, '//form[2]/section/table/tbody/tr/td[3]/input')

    if current\_test\_failed: return

    next\_button.click()

# Reads the text of the results page to see what score out of 30 we got

def get\_score(driver) -> int:

    # Find the header with the score

    header = find\_element(driver, 'id', 'page\_title\_h1\_lbl').text

    return int(header.replace('Practice Scenario Results: Score ', '').replace(' of 30.', ''))

# A version of driver.find\_element that will signal the program to move on to the

# next test instead of throwing an error (now that I think about it I could've just surrounded)

# lines 67 - 68 with try catch and it would've been so much easier)

def find\_element(driver, find\_method, method\_value: str):

    global current\_test\_failed, failed\_tests

    if current\_test\_failed: return

    try:

        return driver.find\_element(find\_method, method\_value)

    except NoSuchElementException:

        # Move to next test

        print('ran into NoSuchElementException, moving on to next test')

        failed\_tests += 1

        current\_test\_failed = True

if \_\_name\_\_ == '\_\_main\_\_':

    main()