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 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

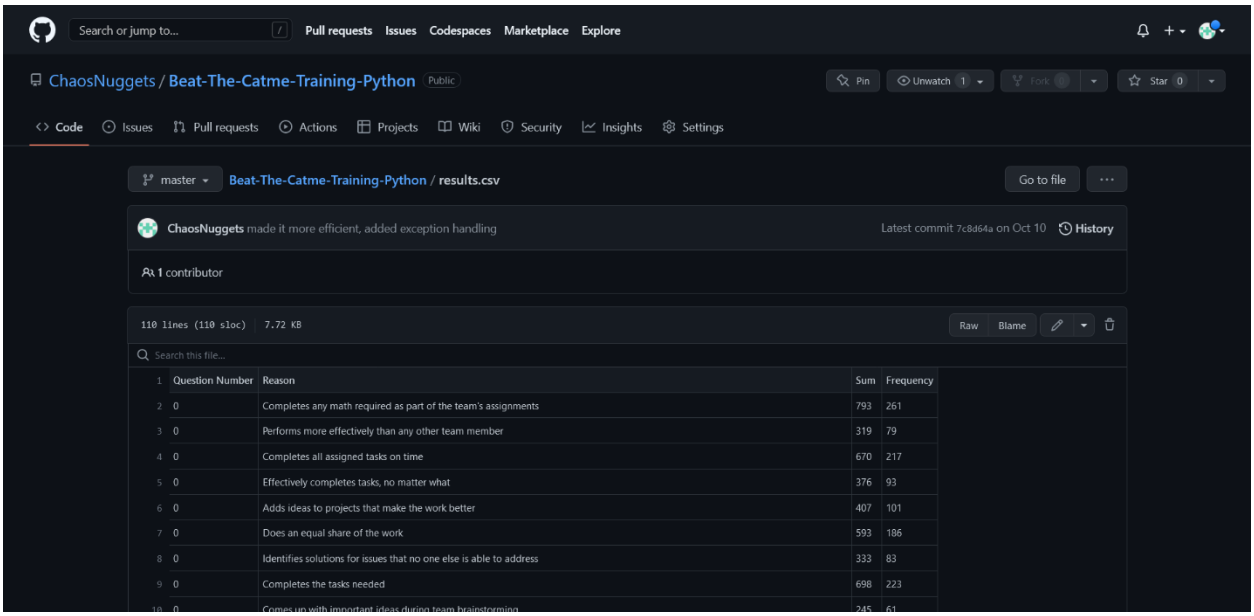
Functions Inside test_algorithm.py

- 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 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, 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

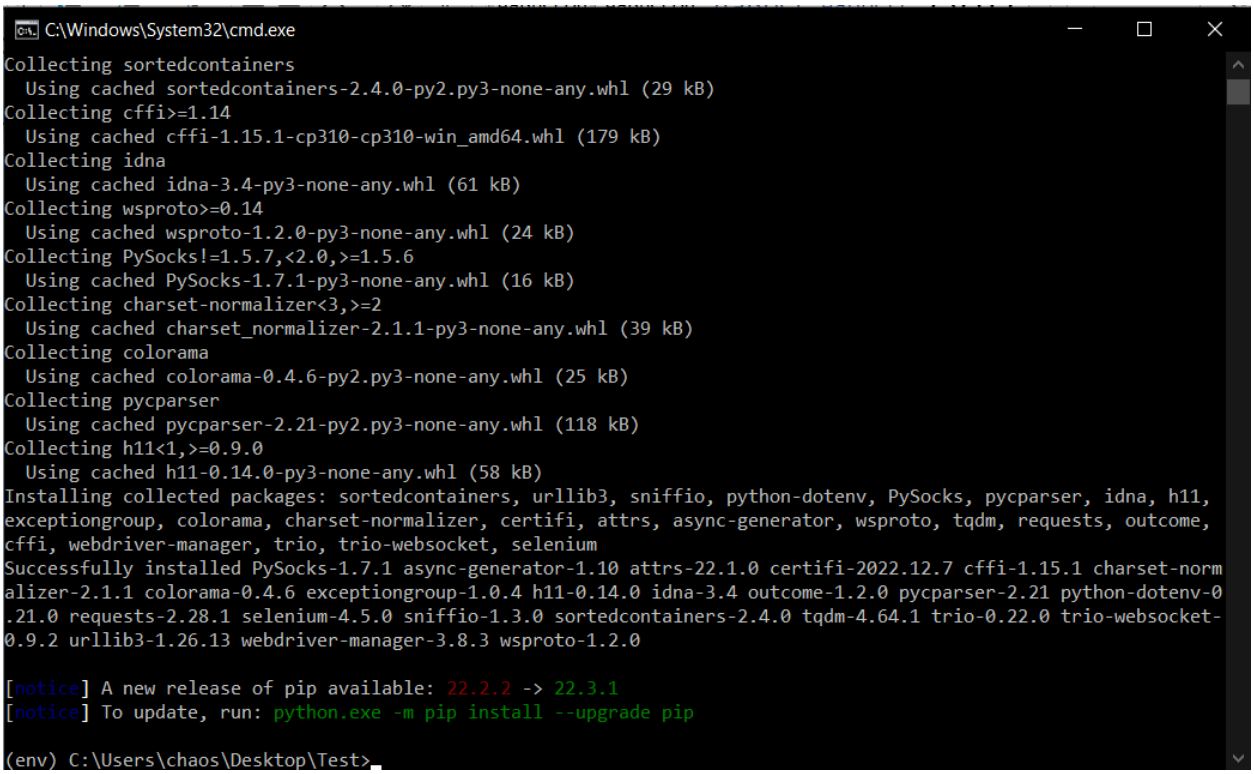
In order to run get_catme_data.py, you need to have the selenium 4.5.0 and webdriver-manager 3.8.3 installed. To install these libraries, run these commands:

pip install selenium

pip install webdriver-manager

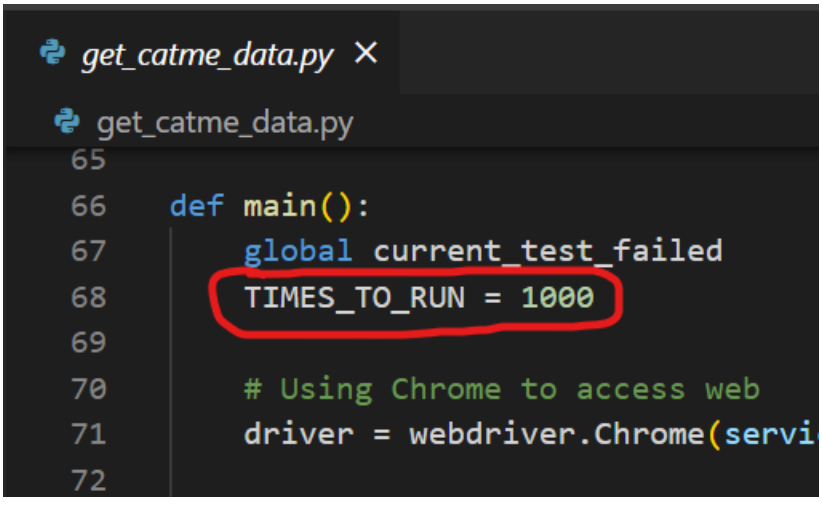
Alternatively you can download requirements.txt from GitHub and run:

pip install -r requirements.txt

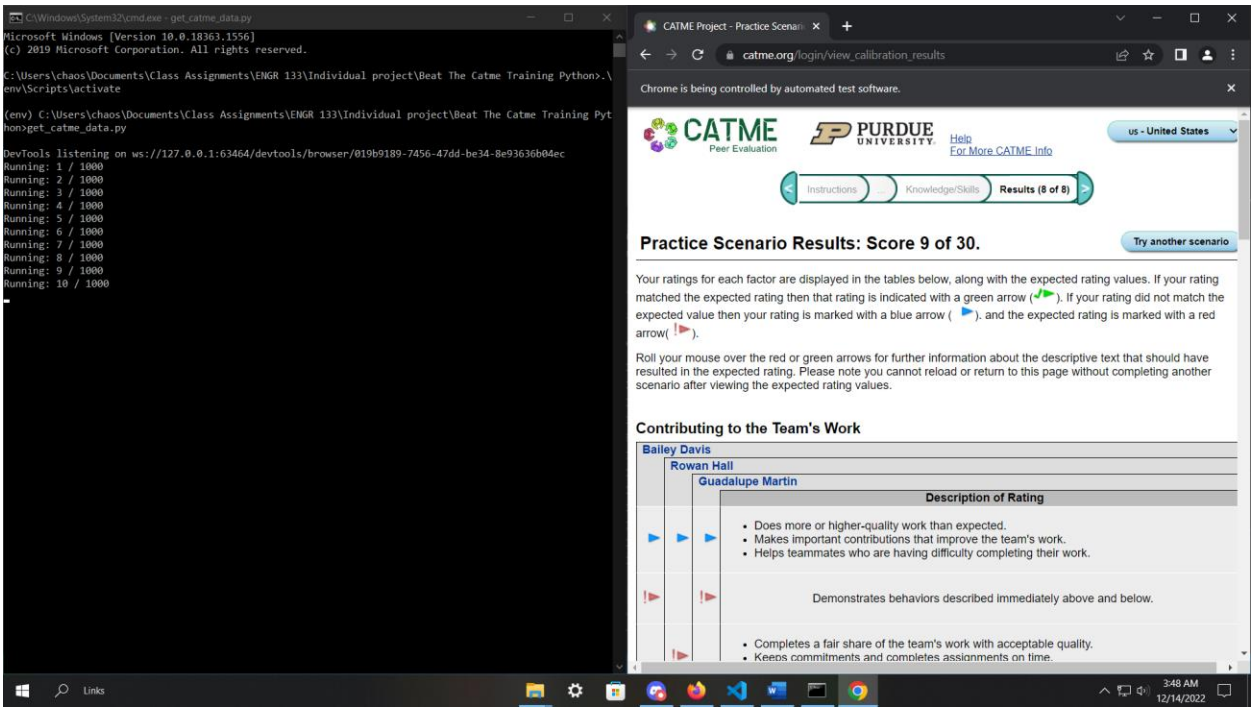


installing requirements with pip

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.



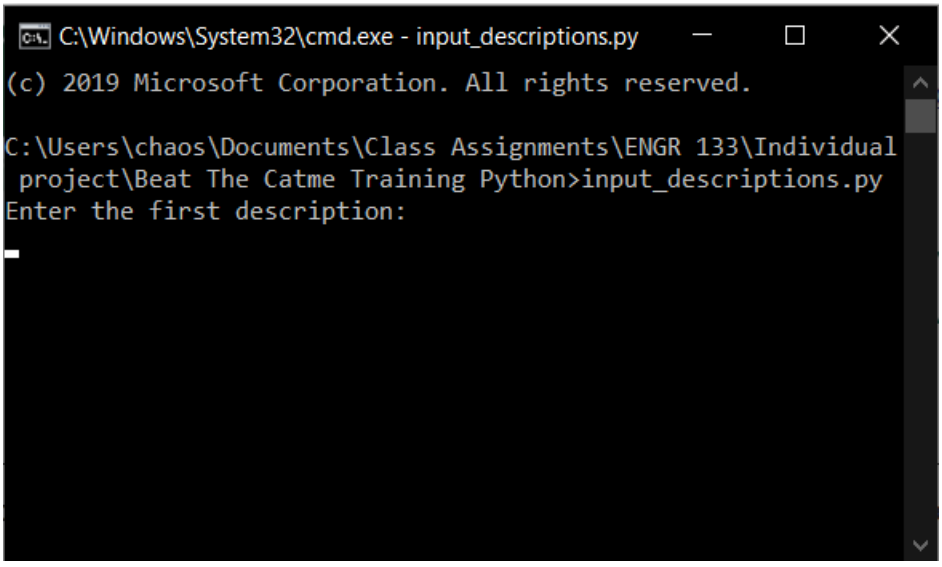
the TIMES_TO_RUN variable inside get_catme_data.py



what happens when you run get_catme_data.py

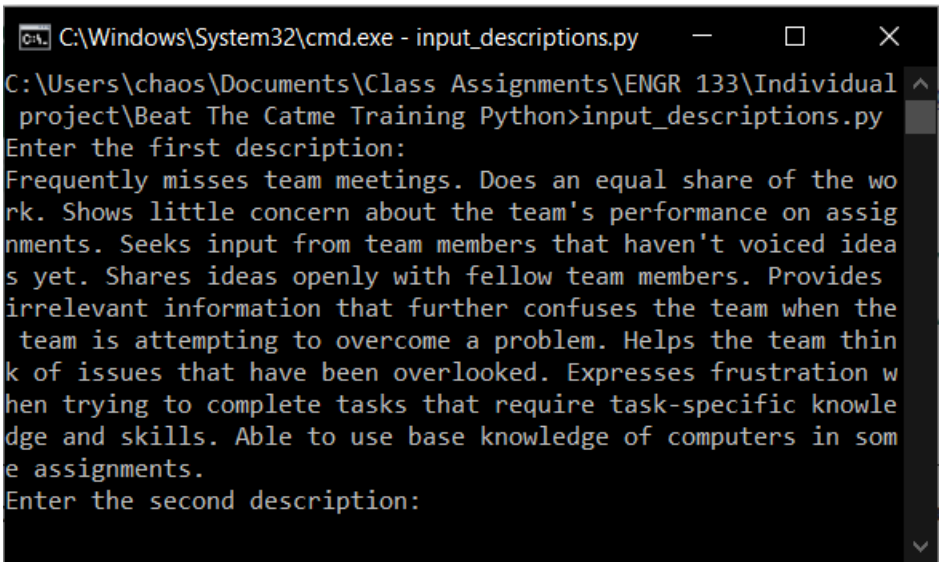
Step 2: Getting the Answers

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:”.



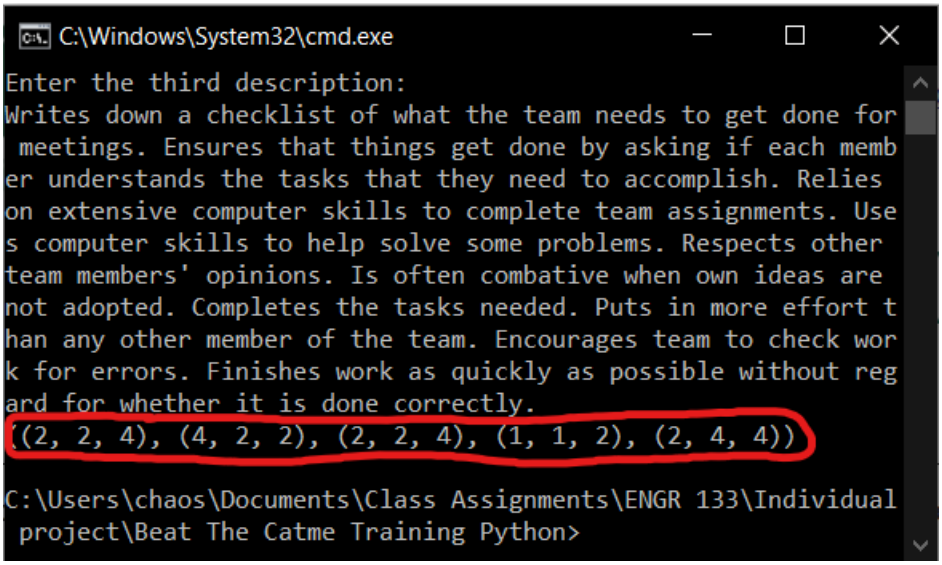
The first thing that will happen when you run input_descriptions.py

Paste the first description and then press enter.



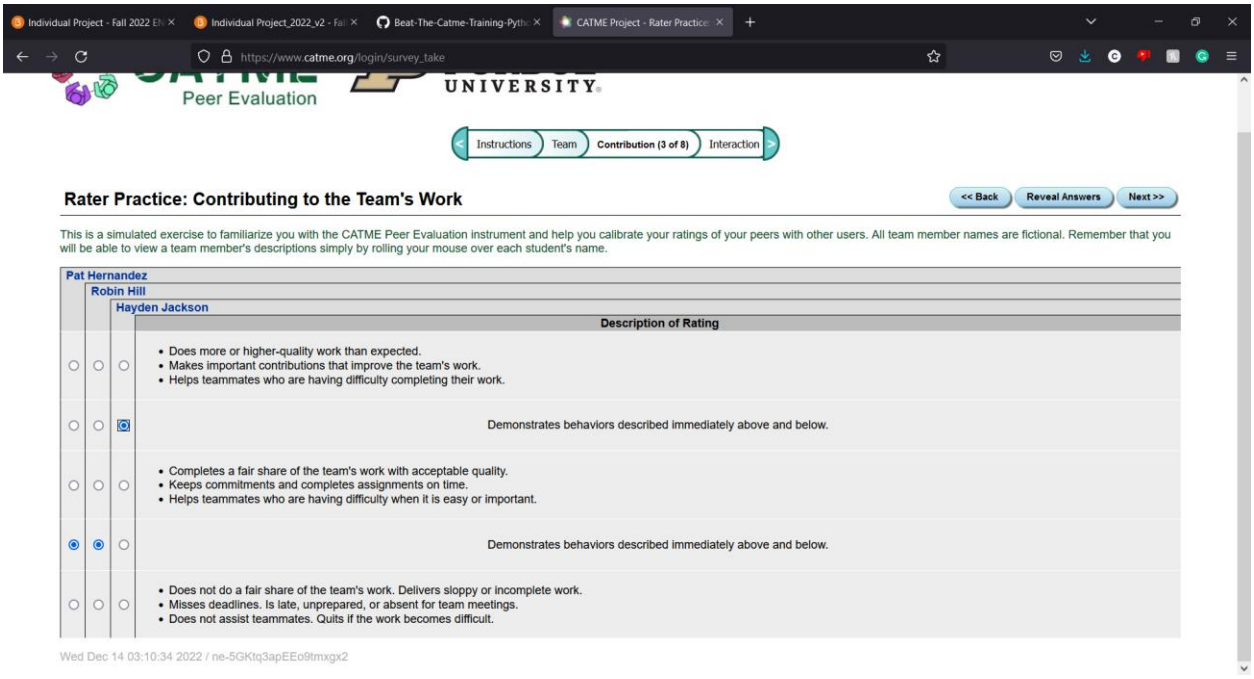
After you enter the first description

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)).



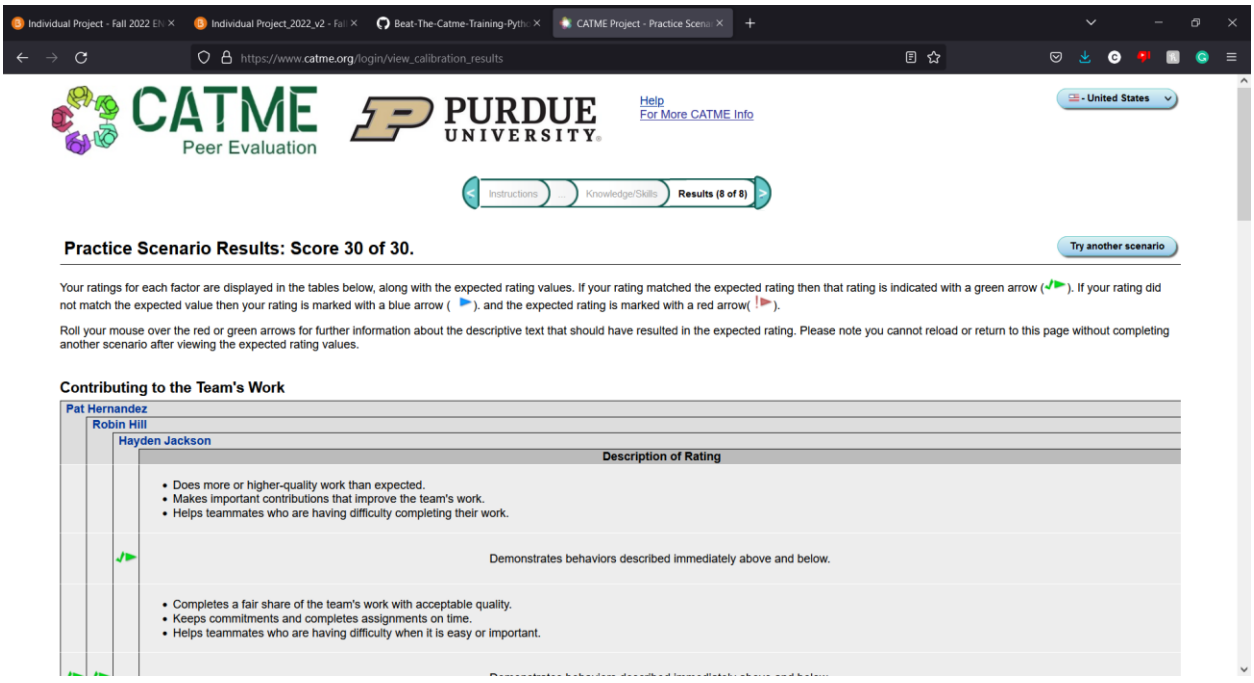
input_descriptions.py answers

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.



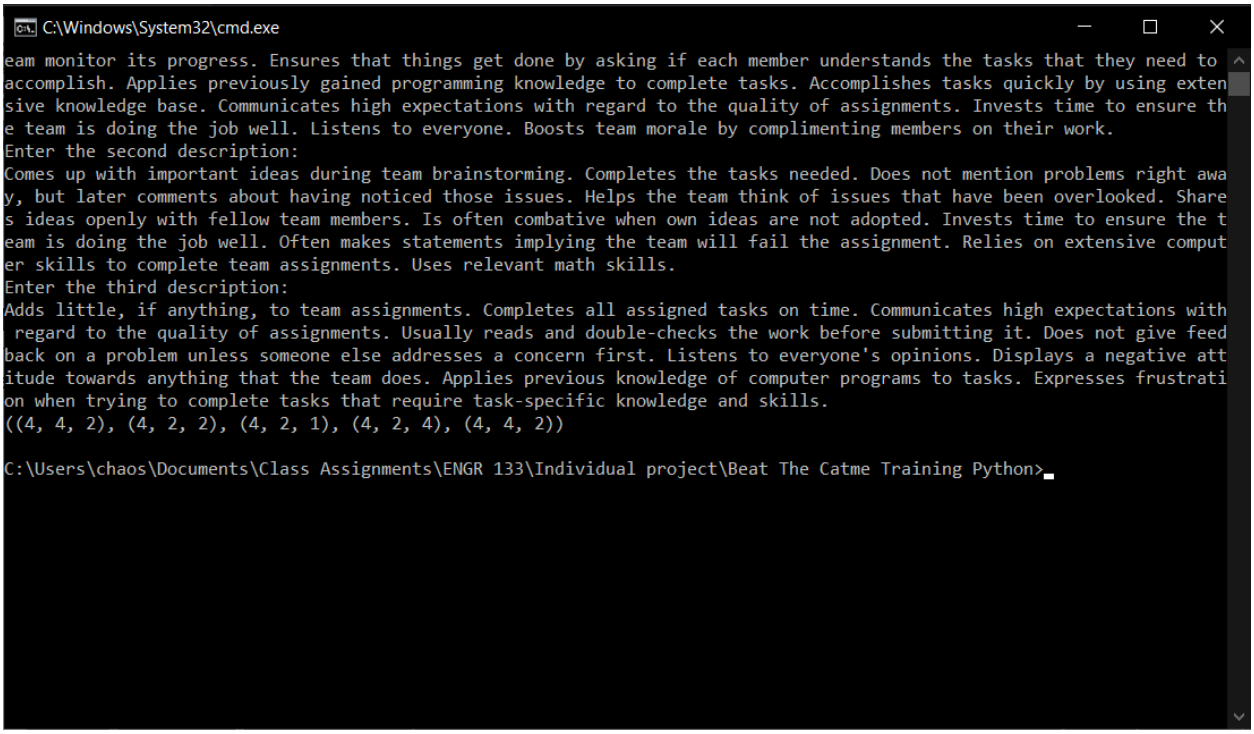
The correct answers for question #1

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

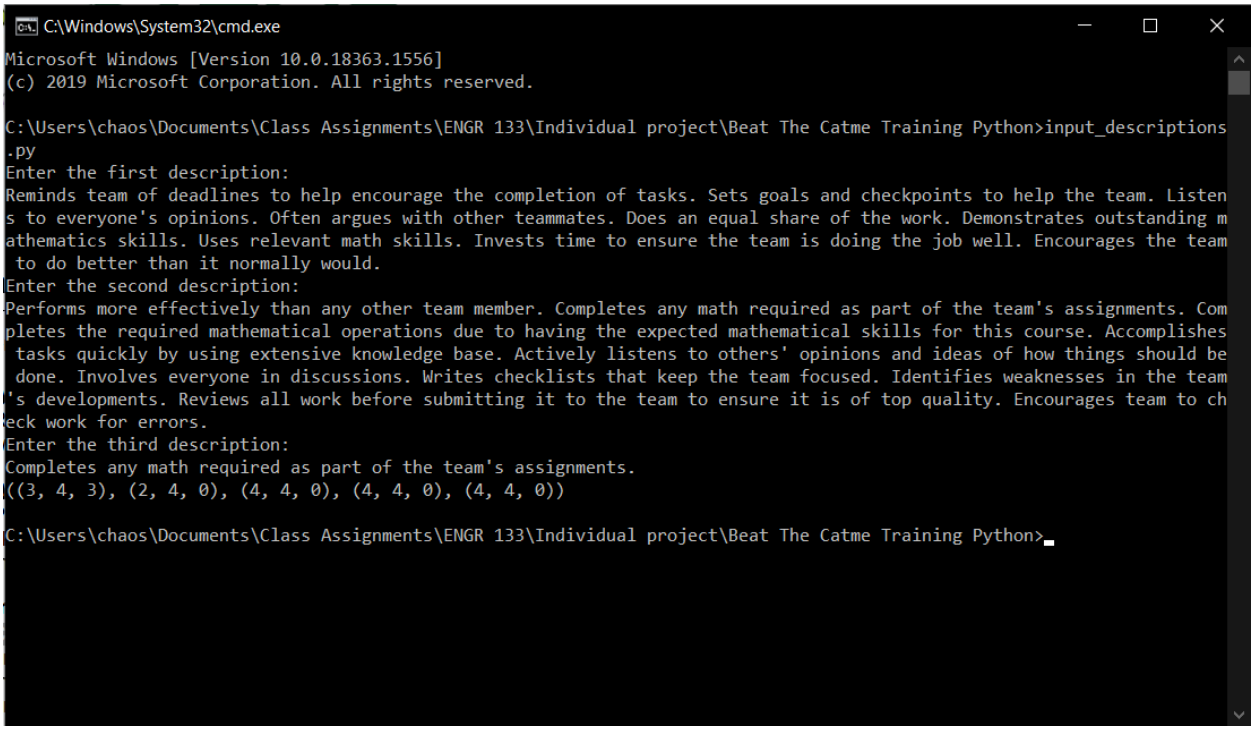


Catme training 30 out of 30 points

Sample Inputs and Outputs



a regular scenario where the user does everything correctly



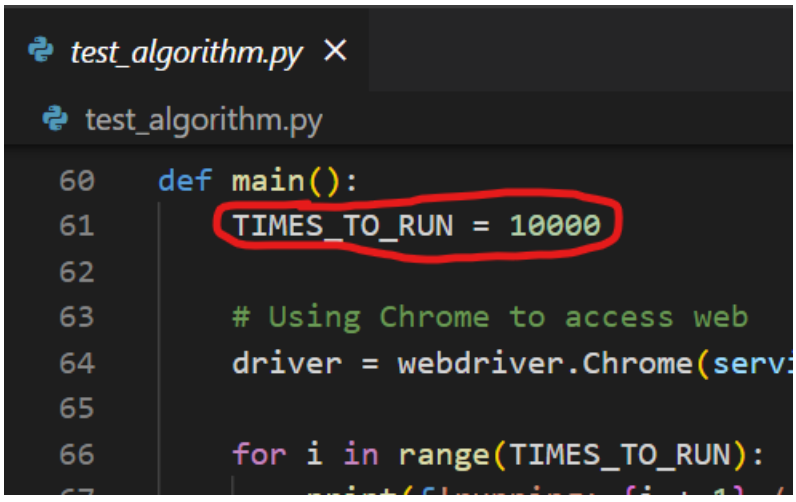
If the user enters and incomplete description, some of the questions don’t have enough information to be answered. This is shown by the 0’s.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python>input_descriptions
.py
Enter the first description:
I wanna sleep
Enter the second description:
it's so late
Enter the third description:
I'm very tired
Traceback (most recent call last):
  File "C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python\input_desc
riptions.py", line 50, in <module>
    main()
  File "C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python\input_desc
riptions.py", line 45, in main
    answers = calculate_answers.calculate_answers(descriptions)
  File "C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python\calculate_
answers.py", line 47, in calculate_answers
    answers.append(calculate_paragraph_ratings(description, NUMBER_OF_QUESTIONS))
  File "C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python\calculate_
answers.py", line 114, in calculate_paragraph_ratings
    raise RuntimeError(f'sentence "{sentence}" could not be found in the data')
RuntimeError: sentence "i wanna sleep" could not be found in the data

C:\Users\chaos\Documents\Class Assignments\ENGR 133\Individual project\Beat The Catme Training Python>
```

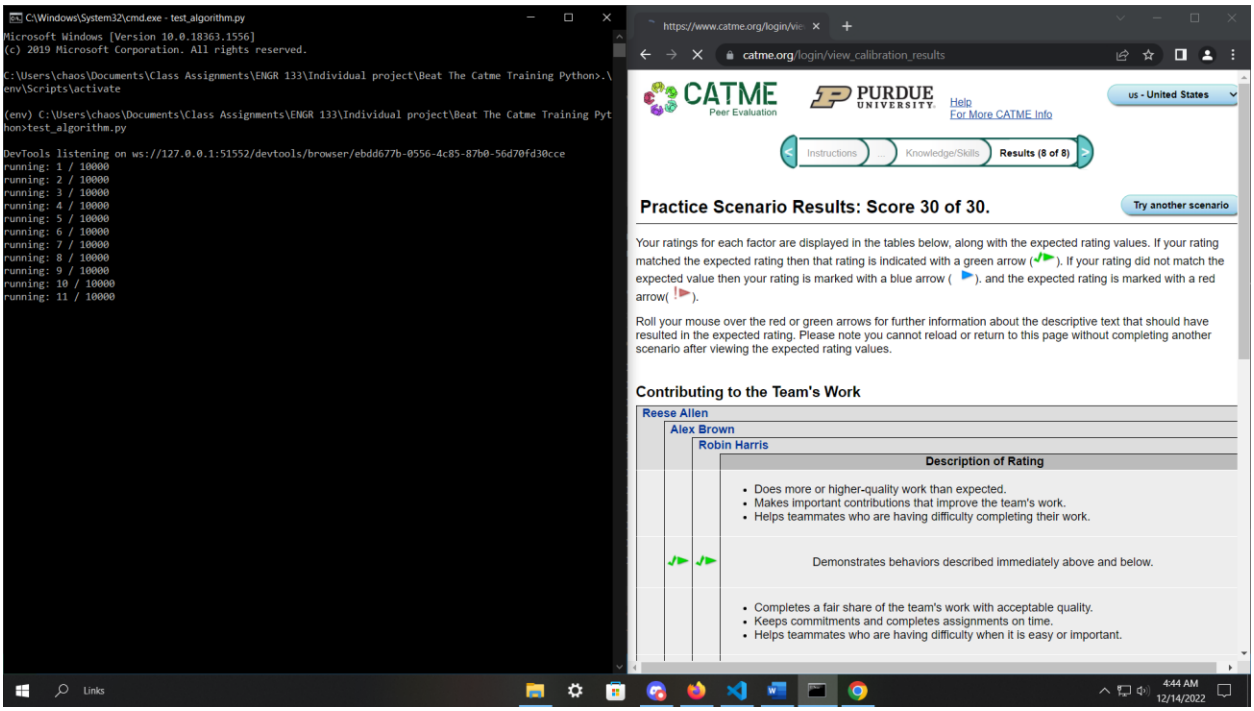
The sentence “I wanna sleep” is not a sentence that would appear in the catme training, so it’s not in our data. The program responds to this by throwing an error saying, “sentence ‘i wanna sleep’ could not be found in the data”.

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
```

```python
# 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()
```