CSE 8A/6R Fall 2025 Homework 6

Due: Thursday, December 4, 2025 before 11:59 pm

Full Name: _____ PID: _____

Homework 6 Topics Covered:
1. Testing 2. Program Comprehension

5 core competencies:
1. Code Tracing, 2. Code Comprehension, 3. Code Writing, 4. Debugging, and 5. Testing

Submission

Questions 1 - 3:
- **Must be handwritten** in the given boxes only.
- Scanned/uploaded to Gradescope **Homework 6 Questions 1 - 3** assignment.
- *If any of the questions 1 - 3 are typed*, *then your work will be considered* ***ungradable****.*

Question 4 (process journal):
- The **process journal** can be either handwritten or **typed (preferred)** into a document.
- **In addition, you'll need to submit the Python file (.py) to the problem. See submission instructions below.**
- Uploaded to Gradescope **Homework 6 Question 4 Process Journal** assignment.

When you submit your work, **please make sure to assign the correct pages to the questions**. Otherwise, your work will be considered **ungradable**.

Point breakdown for Homework 6:
- **Exemplary (3 homework points):** All problems receive a "Pass."
- **Satisfactory (2 homework points):** At least 3 out of 4 questions receive a "Pass."
- **Needs Improvement (1 homework point):** At least 1 out of 4 questions receives a "Pass."
- **Incomplete (0 homework points):** No problems receive a "Pass," or the assignment was not submitted, or the assignment is ungradable (e.g., illegible handwriting), or if the submission does not follow the instructions (e.g., homework is typed instead of handwritten, not answering in the boxes provided, not assigning pages to the question).

1. [Testing, Code Writing] Use the **"0-1-many" pattern in the reading** to come up with **three tests** each for the following programs. You don't need to show the resulting outputs.

A) Radioactive decay.

```
# t is a time in seconds, as a float value.
def radioactive_decay(t):
    N0 = 1000      # initial quantity, fixed
    lam = 0.00012  # decay constant (per second)

    # Approximate e^x
    x = -lam * t
    exp_approx = 1 + x + (x**2)/2 + (x**3)/6 + (x**4)/24 + (x**5)/120

    # Decay formula: N = N0 * e^(-lam * t)
    N = N0 * exp_approx
    return N
```

Test 1:
```
print(radioactive_decay(_____))
```

Test 2:
```
print(radioactive_decay(_____))
```

Test 3:
```
print(radioactive_decay(_____))
```

B) Find the index of the last occurrence of an `item` in a `target_list`. Returns the index of the last occurrence of the item, or None if `item` is not found in the `target_list`.

```
def last_occur(target_list, item):
    last_index = None
    for i in range(len(target_list)):
        element = target_list[i]
        if element == item:
            last_index = i
    return last_index
```

Test 1:
```
print(last_occur(_____))
```

Test 2:
```
print(last_occur(_____))
```

Test 3:
```
print(last_occur(_____))
```

C) Get capital by country. For this part, please write your own function calls similar to how we implement tests for parts A and B, still following the “0-1-many” pattern from readings.

```
# info is a dict where keys are country names and values are capitals
# both the key and value of the dict are strings
# country is a string representing the country to search for
def get_capital(info, country):
    for name, capital in info.items():
        if name == country:
            return capital
    return None
```

Test 1:

Test 2:

Test 3:

2. [Testing, Code Writing] For the following program, explain how many tests are needed to cover all paths and come up with those tests. You may leave additional tests blank (for example, if you say 2 tests are needed to cover all paths, you only need to create test #1 and test #2). For each test case, write out the expected outputs. You don't need to show the actual outputs.

A) Classifying animals

```
# num_legs is a non-negative integer.
# meows is a boolean value.
def classify_animal(num_legs, meows):
    if num_legs == 4:
        if meows:
            return "Cat"
        else:
            return "Dog"
    elif num_legs == 2:
        return "Human"
    return "Unknown"
```

Number of tests needed: _____

Explain your answer:

Test 1: `print(classify_animal(_____))`

`# Test 1 expected output: _____`

Test 2: `print(classify_animal(_____))`

`# Test 2 expected output: _____`

Test 3: `print(classify_animal(_____))`

`# Test 3 expected output: _____`

Test 4: `print(classify_animal(_____))`

`# Test 4 expected output: _____`

Test 5: `print(classify_animal(_____))`

`# Test 5 expected output: _____`

Test 6: `print(classify_animal(_____))`

`# Test 6 expected output: _____`

B) Trip idea

```
# season and budget are strings.
def trip_idea(season, budget):
    if season == "winter":
        destination = "the mountains"
    else:
        destination = "the beach"
    if budget == "high":
        style = "luxury"
    else:
        style = "budget"
    return f"A {style} trip to {destination} sounds fun!"
```

Number of tests needed: _____

Explain your answer:

Test 1: `print(trip_idea(_____))`

`# Test1 expected output: A _____ trip to _____ sounds fun!`

Test 2: `print(trip_idea(_____))`

`# Test2 expected output: A _____ trip to _____ sounds fun!`

Test 3: `print(trip_idea(_____))`

`# Test3 expected output: A _____ trip to _____ sounds fun!`

Test 4: `print(trip_idea(_____))`

`# Test4 expected output: A _____ trip to _____ sounds fun!`

Test 5: `print(trip_idea(_____))`

`# Test5 expected output: A _____ trip to _____ sounds fun!`

3. [Program Comprehension] Read the following code and answer the questions below.

```
def get_min_value(some_list):
    min_value = some_list[0]
    for value in some_list:
        if value < min_value:
            min_value = value
    return min_value

# assumes input_list is a list of integers.
def mystery(input_list):
    result_list = []
    while len(input_list) > 0:
        current_min = get_min_value(input_list)
        result_list.append(current_min)
        input_list.remove(current_min)
    return result_list
```

Explain in plain English the purpose of the mystery function. As a reminder, to explain a mystery function in plain English, the key is to use a couple of words that describe the high-level function behaviors. I.e., If you were to rename the function to something meaningful, how would you name it?

4. [Program Comprehension, Code Writing]

For this question, you are submitting a ***process journal***. For detailed instructions, please check the instruction document: https://tinyurl.com/ProgrammingProcessJournal. IMPORTANT: Please submit your python file (.py) (in addition to your process journal) that you used to implement the program. Submission instructions are now attached to the document above.

During lecture 16, we talked about program comprehension using the example of course grade calculator. For this question, you should use the following base code: [link to base code].

Below are two tasks you should complete using the base code:

1. Identify an existing issue in the base code. Currently, the way it handles a **failing grade** has some issues. Find the bug and fix it.
2. Implement a new feature. Allow the program to handle the case where low social learning points (i.e., social_points < 12) will result in a **one letter grade reduction** (i.e., if you have fewer than 12 total social points, your base letter grade will be lowered by one full letter, e.g., a B becomes a C). To do this, you may either change the existing functions, or create new function(s).

You must have the following test cases:

- At least **three** test cases to validate that you have successfully debugged the existing issue, and the original calculations (i.e., cases where social_learning_points ends up in either a plus, minus, or no modifier) works correctly. You should ensure that these test cases are still working after you implemented the new feature. This is called "*regression testing*": to check that new code modifications aren't harming existing functionality or introducing any new bugs.
- At least **three additional** test cases to test the newly implemented one letter grade reduction feature.

We attached the grade calculation rules (from the syllabus) on the next page.

Pasted from our course syllabus:

Assigning Letter Grades

Your final letter grade in the course is determined by a two-step process. First, your performance on homework and exams sets your **base letter grade**. Second, your consistent engagement in social learning activities can modify that grade with a plus (+), a minus (-), or even lower it by a full letter.

Step 1: Determine Your Base Letter Grade

Your base grade is determined by meeting the minimum point thresholds for **both** homework and exams. There are 6 homework assignments (18 points max) and 2 exams (6 points max).

- **To earn an A:** You must have at least **15 total homework points** AND **6 total exam points**.
- **To earn a B:** You must have at least **12 total homework points** AND **4 total exam points**.
- **To earn a C:** You must have at least **9 total homework points** AND **2 total exam points**.
- **F:** Awarded if you do not meet the requirements for a C.

Step 2: Apply Social Learning Modifier

Your total social learning points (out of a possible 30) will then adjust your base grade.

- **Plus (+) modifier:** Earned if you have **at least 24 total social points**.
- **No modifier:** Earned if you have **18 to 23 total social points**.
- **Minus (-) modifier:** Earned if you have **12 to 17 total social points**.
- **One letter grade reduction:** If you have **fewer than 12 total social points**, your base letter grade will be lowered by one full letter (e.g., a B becomes a C).

NOTE: No student can earn a D grade in our course. So, if someone's base grade is a C and if they have less than 12 social_points then their final letter grade will be a F (not a D). Also, it is impossible to get a modifier for the F grade since it's indicating failing the class.