Version Santa: 🎅

"Success is the sum of small efforts, repeated day in and day out." – Robert Collier

CSE 8A-6R Fall 2025 Final Exam

Thursday, December 11, 2025 (3pm - 5:59pm)

Full Name: _____ PID: _____

Topics Covered:
1) Variables and Datatypes, 2) Functions, 3) Conditional statements, 4) Sequences: Lists and Strings, 5) Loops: For loops and While Loops, 6) Dictionaries, 7) References and Objects, 8) How Functions Work, Variable Scope, 9) Nested For Loops and 2D lists, 10)Debugging, 11) Testing, 12) Program Comprehension.

***5 Core Competencies* Tested:**
1. Code Tracing, 2. Code Comprehension, 3. Code Writing, 4. Debugging, and 5. Testing

Important Instructions (Please Read Carefully)

- This exam contains **16 pages** with **10 questions total**.
- All questions on the final exam should be handwritten with **legible handwriting**
- You work will be marked **'Ungradable'** if your **handwriting is illegible**
- You should **show your process** of arriving at the answer in the **"show your work"** space
- If you **do not show your work** and only write the final answer, your answer will be marked as **'incorrect'** even if the final answer is correct
- You are **not allowed** to use *note sheets* or *calculators* for this exam
- You are **required to work on this exam on your own**. If we find any signs of **academic misconduct** (e.g., peeking into other people's exam, using a note sheet), we will immediately mark your exam as "No Pass" and **you will be reported to the UCSD's academic integrity office**
- **No questions will be answered during the exam.** If there are any mistakes in any question on the exam, then everyone will get full credit for those questions
- **Write your PID** on *all* the odd pages
- There is a **reference sheet** with useful information at the end of this exam

Exam Grading

Individual questions on this exam will be graded on the following scale:

- **Correct:** The final answer is correct, and the supporting work is logical, follows a valid method, and shows a correct understanding of the concepts.
- **Mostly Correct:** The supporting work demonstrates a strong understanding of the concept and a valid method, but contains minor errors (e.g., a small calculation mistake) that lead to an incorrect final answer.
- **Incorrect:** The supporting work is missing, illogical, or contains significant conceptual errors, regardless of the final answer.

1. [Code Tracing, Code Comprehension]

```
def enigma(happiness_dict):
    result = {}
    for key in happiness_dict:
        value = happiness_dict[key]
        result[value] = key
    return result

happiness_rankings = {"USA": 16, "India": 136, "Mexico": 24, "Finland": 1,
"UK": 18, "New Zealand": 7}

print("happiness_rankings =", happiness_rankings)
answer = enigma(happiness_rankings)
print("answer =", answer)
```

A) What will be printed by the code above?

Show your work

Output:

B) Explain in plain English the high level purpose of the enigma function.

PID: _____

2. [Code Comprehension]

Read the following code and answer the questions below.

```
def helper(some_list):
    return len(some_list) - 1

# numbers is a list of integers
def mystery(numbers):
    result = []
    index = helper(numbers)
    while index >= 0:
        result.append(numbers[index])
        index -= 1
    return result
```

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. If you were to rename the function to something meaningful, how would you name it?

Explain in plain English the purpose of the **`helper`** function. If you were to rename the `helper` function to something meaningful, how would you name it?

3. [Code Tracing]

Predict the output of the following code snippets. **Draw memory diagrams** (i.e., call stack with the stack frames containing references and objects) to explain what is happening behind the scenes in the computer's memory.

A) Adding one to an integer

```
def add_one_to_integer(num):
    num += 1
    return num

num = 42
print("num (before) =", num)
add_one_to_integer(num)
print("num (after) =", num)
```

Output:

Memory diagram:

PID: _____

B) Adding one to a list of integers

```
def add_one_to_list(numbers):
    for i in range(len(numbers)):
        numbers[i] += 1


nums = [11, 3, 14]
print("nums (before) =", nums)
add_one_to_list(nums)
print("nums (after) =", nums)
```

Output:

Memory diagram:

4. [Code Writing]

You are tasked with writing two Python functions to process a list of email addresses. The goal is to first **extract the usernames from the emails** and then **combine those usernames into a single string separated by commas**.

Requirements
Implement `get_usernames(emails)`, which accepts a list of emails (as a list of strings). The function should return a list containing only the usernames (the text before the `@` symbol) from each email, ideally using a loop and the `.split()` method.

Implement `combine_usernames(usernames)`, which accepts a list of username strings. The function should return a single string consisting of all the `usernames` separated by commas, utilizing the `.join()` method.

```
def get_usernames(emails):

    _____________________________________________________________________

    _____________________________________________________________________

    _____________________________________________________________________

    _____________________________________________________________________

    _____________________________________________________________________

def combine_usernames(usernames):

    _____________________________________________________________________

emails = ["[redacted]", "[redacted]", "[redacted]"]
print(get_usernames(emails))
# Expected output: ['[redacted]', '[redacted]', '[redacted]']

print(combine_usernames(get_usernames(emails)))
# Expected output: '[redacted],[redacted],[redacted]'
```

PID: _____

5. [Testing]

Write **five (5) distinct test cases** for the function below.

Function Description: The function `count_matches(nums, target)` iterates through a list of integers (`nums`) and returns an integer representing how many times `target` (integer) appears in the list.

```
def count_matches(nums, target):
    total = 0
    for num in nums:
        if num == target:
            total += 1
    return total
```

Fill in the function calls below to test the specific scenarios described and write the expected output for each test case.

Expected format of the test cases for full credit:

```
print(count_matches([12, 42, 11], 42))    # Expected output: 1
```

Test 1: Scenario: The list passed to the function is **empty**.

_____

Test 2: Scenario: The list is **not** empty, but the `target` is NOT found anywhere in the list.

_____

Test 3: Scenario: The list contains exactly **one item**, and that item matches the `target`.

_____

Test 4: Scenario: The list contains multiple items, but the `target` appears **exactly once**.

_____

Test 5: Scenario: The list contains multiple items and the `target` appears two or more times in the list.

_____

6. [Debugging]

You have a theater seating chart represented by a 2D list, where 0 is available and 1 is taken.

You are writing a function `book_seat` that takes the grid, a row index, and a column index. It should check if the specific seat is available (0). If it is, it should update that seat to be taken (1) and return `True`. If it is already taken, it should return `False`.

However, the booking system is failing to actually reserve the seats.

```
def book_seat(seats, row_idx, col_idx):
    # Access the specific seat at the given row and column
    seat_status = seats[row_idx][col_idx]

    if seat_status == 0:
        seat_status = 1  # Mark the seat as booked
        return True
    else:
        return False

# Initial State: All seats are empty (0)
theater = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]

# Attempt to book the middle seat (Row 1, Column 1)
success = book_seat(theater, 1, 1)

if success:
    print("Booking successful!")
else:
    print("Booking failed - seat already taken.")

print("Theater State:")
for row in theater:
    print(row)
```

Expected Output:	Actual Output:
Booking successful! Theater State: [0, 0, 0] [0, 1, 0] [0, 0, 0]	Booking successful! Theater State: [0, 0, 0] [0, 0, 0] [0, 0, 0]

PID: _____

A) **Identify the Bug:** Explain why the `theater` list remains unchanged even though the function returned `True` and the line `seat_status = 1` was executed.

B) **Fix the Code:** Rewrite the `book_seat` function so that it successfully updates the 2D list.

```
def book_seat(seats, row_idx, col_idx):
    # Your corrected function definition (i.e., implementation) goes here
```

7. [Code Writing]

You are analyzing sales data for a chain of coffee shops. You have a 2D list called `sales_data`, where:

- Each row represents a specific store location (e.g., Downtown, Campus, Beachside).
- Each column represents the sales in dollars made during a specific hour of the day.

Task: Write a function `calculate_total_revenue(sales_grid)` that takes the 2D list as input and returns a **single integer** representing the sum of all sales from all stores combined.

Note: You may assume that all sales data entries are integers.

```
sales_data = [
   [10, 20, 50], # Downtown store sales
   [5, 5, 10],   # Campus store sales
   [100, 0, 0]   # Beachside store sales
]

def calculate_total_revenue(sales_grid):









# Calculate the total revenue from all stores
total_revenue = calculate_total_revenue(sales_data)
print("Total revenue from all stores:", total_revenue)
# Expected output: 200
# (Explanation: 10 + 20 + 50 + 5 + 5 + 10 + 100 + 0 + 0 = 200)
```

PID: _____

8. [Debugging, Code Writing]

Consider the following Python code. The intended behavior is for the function to return "Found" if the specified name is present in the TA list, and "Not Found" otherwise.

```
def is_ta_present(name, ta_list):
    for i in range(len(ta_list)):
        if ta_list[i] == name:
            return "Found"
        else:
            return "Not Found"

ta_list = ['Andrew', 'Hailey', 'Liam', 'Prudhviraj', 'Sang']

# Test Case 1
print(is_ta_present('Andrew', ta_list))
# Expected output: Found
# Actual output: Found

# Test Case 2
print(is_ta_present('Sang', ta_list))
# Expected output: Found
# Actual output: Not Found
```

A) **Identify the bug** in the function. Explain why the actual output does not match the expected output for Test Case 2.

B) **Rewrite the function** to fix the bug such that the function works as expected.

9. [Testing]

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.

Ideas for a Movie Date!

```
# genre and time_of_day are strings.
def movie_date_idea(mood, time_of_day):
  if mood == "romantic":
      genre = "romantic comedy"
  else:
      genre = "horror"
  if time_of_day == "morning":
      setting = "at AMC"
  elif time_of_day == "afternoon":
      setting = "at a rooftop"
  else:
      setting = "in the living room"
  return f"Let's watch a {genre} movie {setting}!"
```

Number of tests needed: _____

Explain your answer:

PID: _____

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

# Expected: Let's watch a _____ movie _____!
```

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

# Expected: Let's watch a _____ movie _____!
```

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

# Expected: Let's watch a _____ movie _____!
```

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

# Expected: Let's watch a _____ movie _____!
```

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

# Expected: Let's watch a _____ movie _____!
```

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

# Expected: Let's watch a _____ movie _____!
```

Test 7:
```
print(movie_date_idea(_____))

# Expected: Let's watch a _____ movie _____!
```

Test 8:
```
print(movie_date_idea(_____))

# Expected: Let's watch a _____ movie _____!
```

10. [Code Tracing, Debugging]

Consider the function below that calculates the social learning grade for a week in CSE 8A/6R. A student's code for their homework 2 is below. We found a bug in its implementation.

```
1  def social_learning_grade(reading, lectures, lab, study_group):
2      total_passes = reading + lectures + lab + study_group
3      if total_passes >= 4 and lab == 1 and reading == 1:
4          return 3  # Exemplary
5      elif total_passes >= 3 and lab == 1 or reading == 1:
6          return 2  # Satisfactory
7      elif total_passes >= 2:
8          return 1  # Needs Improvement
9      else:
10         return 0  # Incomplete
11
12 print(social_learning_grade(1, 0, 0, 0))
```

Expected Output:

```
0
```

Actual Output:

```
2
```

A) Explain why the actual output is different from the expected output.

B) Explain how we can fix the bug. You do not need to rewrite the function.

PID: _____

Reference Sheet #1

Some common things that you may want to know or use.

`range(stop)` where `stop` is an integer	Returns a list of the integers between 0 and `stop`, including 0 but not including `stop`.
`range(start, stop)` where `start` and `stop` are integers	Returns a list of the integers between `start` and `stop`, where `start` is included but `stop` is not.
`range(start, stop, step)` where `start`, `stop`, and `step` are integers	Returns a list of the integers starting at `start`, incrementing (or decrementing) by `step` at each interval, and continuing until the sequence reaches or passes `stop`. The `start` value is included, but the `stop` value is not included.
`len(my_list)`	Returns length of list `my_list`

`<  <=`
`>  >=`
`==  !=`
	Comparison operators for comparing two values to yield a Boolean value (`True` or `False`). (We expect you to know how to use them and how each works.)

`and`
`or`
`not`
	Operators to combine one or two Boolean values to produce a single Boolean value.

`my_list[a:b]` where `my_list` is a list, and `a` and `b` are integers	Returns a **slice** of the list from `my_list[a]` (inclusive) to `my_list[b]` (exclusive).
`item` **`in`** `sequence`	The **`in`** operator checks for the presence of an `item` within a `sequence` (like a list, string). It returns a boolean value (`True` if present, `False` otherwise).
`random.randint(a, b)`	The `random.randint(a, b)` function returns a random integer `N` such that: `a <= N <= b`
`my_string.split(sep)`	`my_string.split(sep)` where `my_string` is a string and `sep` is a string. Returns a list of substrings from `my_string` using `sep` as the delimiter (or separator).
`sep.join(my_list)`	`sep.join(my_list)` where `sep` is a string and `my_list` is a list of strings. Returns a single string created by concatenating the elements of `my_list`, inserting the `sep` string between each element.

Reference Sheet #2

Operator Precedence in Python

Precedence	Operators	Category	Associativity
1 (Highest)	`()`	Grouping/Parentheses	N/A
2	`**`	Exponentiation	Right-to-Left
3	`*, /, //, %`	Multiplication/Division (Arithmetic)	Left-to-Right
4	`+, -`	Addition/Subtraction (Arithmetic)	Left-to-Right
5	`not`	Logical NOT	Right-to-Left
6	`and`	Logical AND	Left-to-Right
7 (Lowest)	or	Logical OR	Left-to-Right

"How lucky am I to have something that makes saying goodbye so hard."