Version Pumpkin: 🎃

"You have walked the path, step by step. Trust your footsteps now."

CSE 8A-6R Fall 2025 Midterm Exam

Thursday, October 30, 2025 (8pm - 9:50pm)

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

***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 **10 pages** with **6 questions total**.
- All questions on the midterm 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

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] The following code snippet on the left produces the output on the right. Answer the questions that follow.

```
 1  print("line A")
 2
 3  def zero(x):
 4      print("line B")
 5      num_zero = x * 0
 6      print("line C")
 7      return num_zero
 8
 9  def square(y):
10      print("line B")
11      num_squared = y * y
12      return num_squared
13      print("line C")
14
15  print("line D")
16  result_1 = square(3)
17  print("line E")
18  result_2 = zero(4)
19  print("line F")
20  print("result_1:", result_1)
21  print("result_2:", result_2)
```

Output:

```
line A
line D
line B
line E
line B
line C
line F
result_1: 9
result_2: 0
```

A) Why is "`line B`" printed twice?

B) Why is "`line C`" only printed once?

PID: _____

2. [Code Comprehension, Testing] In this question, your goal is to understand what the mystery function does.

```
def mystery(my_list, num):
    for index in range(len(my_list)):
        if my_list[index] == num:
            return index
    return -1
```

A) Below are six tests of the `mystery` function.
For each test, **predict the output.**

Test #1: `print(mystery([11, 42, 14], 42))`

Output #1: _____

Test #2: `print(mystery([-20, -10, 0, 10, 20, 0], 0))`

Output #2: _____

Test #3: `print(mystery([12, 3, 21, 30, 28], 12))`

Output #3: _____

Test #4: `print(mystery([100, 1, 30, 90], 5))`

Output #4: _____

Test #5: `print(mystery([55, 82, 88, 89], 89))`

Output #5: _____

Test #6: `print(mystery([], 42))`

Output #6: _____

B) Explain in plain English the purpose of the mystery function.

3. [Debugging, Code Writing]

A café uses `classify_coffee(ounces)` to classify coffee cups based on how many ounces of coffee they contain. The function contains some errors (bugs) and so sometimes this function works correctly as expected but sometimes it does not work as expected! :(

```
# The function classifies coffee cups based on how many ounces they contain.
# Intended classification logic:
# ounces <= 0:          size → "No Coffee"
# 1 <= ounces <= 5:     size → "Small"
# 6 <= ounces <= 12:    size → "Medium"
# ounces > 12:          size → "Large"

# NOTE: Assume ounces will always be an integer.
def classify_coffee(ounces):
    size = None
    if ounces <= 0:
        size = "No Coffee"
    if ounces >= 1 and ounces <= 5:
        size = "Small"
    if ounces >= 6 and ounces <= 12:
        size = "Medium"
    else:
        size = "Large"
    return size


# Test Case 1: print(classify_coffee(0)) - TEST FAILS
# Expected output: "No Coffee"
# Actual output from buggy code: "Large"

# Test Case 2: print(classify_coffee(4)) - TEST FAILS
# Expected output: "Small"
# Actual output from buggy code: "Large"

# Test Case 3: print(classify_coffee(8)) - TEST PASSES
# Expected output: "Medium"
# Actual output from buggy code: "Medium"

# Test Case 4: print(classify_coffee(16)) - TEST PASSES
# Expected output: "Large"
# Actual output from buggy code: "Large"
```

PID: _____

Part 1: Identify and Explain the Logical Bugs
Analyze the *conditional statement* in the above function and answer the questions below.

A) Explain: Why does test case 1 fail?

Part 2: Fix the Code
B) Rewrite the entire function below. Your corrected code must be logically sound and should use the correct conditional statement(s) to pass all stated classification logic and test cases.

```
def classify_coffee(ounces):
    # Your corrected function definition (i.e., implementation) goes here
```

4. [Code Tracing, Code Comprehension] Consider the following code and an attempt at running the code

```
def analyze_rna():
    """
    RNA strings are composed of letters: A, U, G, and C.
    A codon is a sequence of three letters in an RNA string.
    """
    rna = "AUGAGUAGCUAGAAGCAA"
    stop_codons = ["UAA", "UAG", "UGA"]
    print(f"Let's analyze our RNA string: {rna}")

    # Initialize start and end index
    start = 0
    end = 3

    while True:
        codon = rna[start:end]
        print(f"Current codon is: {codon}")

        if codon in stop_codons:
            print("This is a stop codon. Stop!")
            break

        start += 3   # Adjust start index
        end += 3     # Adjust end index

# Run analysis
analyze_rna()
```

Output:

```
Let's analyze our RNA string: AUGAGUAGCUAGAAGCAA
Current codon is: AUG
Current codon is: AGU
Current codon is: AGC
Current codon is: UAG
This is a stop codon. Stop!
```

PID: _____

A) Show how the three variables `start`, `end`, and `codon` in the function `analyze_rna()` change when we run the code to produce the output above.

B) How does the `analyze_rna` procedure stop running?

C) Explain in plain English (you can also draw, if that is helpful): how does this function go through the given RNA string using the variables `start` and `end`?

5. [Code Writing] Write a function that calculates and returns the area of a triangle when given base and height as its parameters.

The formula for calculating the area of a triangle is:

$$\text{Area} = 1/2 \times \text{base} \times \text{height}$$

A) Identify the properties of your function.

Name of the function:

Parameter(s):

Data type of return value:

B) Write your code in the space provided below.

C) Below, write two tests for your function. *Show your work* for the expected output calculation.

Test #1: _____

Expected output: _____

Actual output: _____

Test #2: _____

Expected output: _____

Actual output: _____

PID: _____

6. [Code tracing, Code comprehension] Consider the following code.

```
candies = {}

candies["KitKat"] = 4
candies["Twix"] = 1
candies["Starburst"] = 10
candies["KitKat"] = 5

print(candies)
```

A) What does the dictionary `candies` look like after executing our code above? *Show your work.*

B) What will be printed when the code below is executed?

```
for candy in candies:
    print(candy)
```

Output:

C) What will be printed when the code below is executed?

```
for candy in candies:
    print(candies[candy])
```

Output:

Reference Sheet

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`

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

"If you can meet with Triumph and Disaster and treat those two impostors just the same"

– Rudyard Kipling