COMP 1023 Final Exam - Fall 2025 - HKUST

Date: December 12, 2025 (Friday)
Time Allowed: 3 hours, 8:30–11:30 am
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **8** questions on **18** pages (including this cover page, appendix and 4 blank pages at the end). You may tear off the appendix and the blank pages at the back.
3. Write your answers in the space provided in the answer book using black/blue ink. *NO pencils please, otherwise you are not allowed to appeal for any grading disagreements.*
4. All programming code in your answers must be written in Python, NumPy, Pandas, and Matplotlib versions as taught in class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional classes, helper functions (including inner functions) and use global variables, nor any library functions not mentioned in the questions.
6. **Type hinting does not need to be included in your code.**
7. No electronic devices are allowed, except for an HKEAA approved calculator.

Student Name	SOLUTIONS & MARKING SCHEMES		
Student ID			
Venue	Sports Hall / LG4 / Rm 2407 (Circle the venue)	Seat Number	

Problem 1 [10 points] True/False Questions

Solution:

Question	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
Answer	F	F	T	F	F	T	T	F	F	F

Marking Scheme:

- 1 point for each correct answer (10 points in total).

Problem 2 [10 points] Branching and Looping

Solution:

(a) Score: 85 Qualified
Score: 58 Unqualified
Score: 72 Qualified
Score: 60 Qualified

(b) `score < 60`

(c)
```
assignment_scores: list[int] = [85, 58, 72, 60]
print(f"Total assignment score: ", end='')

# Initialize total sum and loop control variable
total_sum: int = 0
count: int = 0

# While loop: iterate 4 times (count from 0 to 3, corresponding to 4 scores)
while count < 4:
    total_sum += assignment_scores[count]  # Add current score to total
    count += 1  # Update loop control variable to avoid infinite loop

# Print total sum
print(total_sum)
```

(d)
```
assignment_scores: list[int] = [85, 58, 72, 60]
print(f"Highest assignment score: ", end='')

# Initialize highest_score with the first score in the list
highest_score: int = assignment_scores[0]

# For loop to traverse all scores
for score in assignment_scores:
    # Branching: compare current score with highest_score
    if score > highest_score:
        highest_score = score  # Update highest_score if current score is larger

# Print highest score
print(highest_score)
```

Marking Scheme:

(a) 0.5 points per correct line, 2 points in total.

(b) 1 point

(c)
- 1 point for correct initialization (`total_sum = 0` and `count = 0`).
- 0.5 points for correct while loop condition (`count < 4`).
- 1.5 points for correct loop body (`total_sum += assignment_scores[count]`).
- 0.5 point for correct update (`count += 1`).
- 0.5 point for correct print statement.

(d)
- 1 point for correct initialization of `highest_score` (e.g., `highest_score = assignment_scores[0]`). Hard coding is only acceptable when the `highest_score` is set to a value less than 0 (0 included); otherwise, no points will be given.
- 0.5 points for correct for loop (traverses all scores in `assignment_scores`).
- 1 point for correct branching (`if score > highest_score: highest_score = score`). No points will be given for hard coding. Any function is not allowed.
- 0.5 points for correct print statement (outputs Highest assignment score). If extra words are printed, deduct 0.25 points.

Problem 3 [15 points] Collections

Solution:

(a) (i)
```
Subset: [20, 40, 60]
Popped: 70
Final Data: [50, 60, 80]
```

(ii) (I) Line 4: locations[home_coords] = "Home"

(II) Reason: home_coords is a list, which is a mutable type. Dictionary keys must be immutable (hashable). Python raises a TypeError: unhashable type: 'list'.

(III) Fix: Convert the list to a tuple (which is immutable). locations[tuple(home_coords)] = "Home" OR change definition to home_coords = (22, 114)

(b) (i)
```
[88.0, 95.0]
90.0
False
89.0
```

(ii) (I)
```
[score * 1.1 for score in score_list if score > 85]
```

(II)
```
student_dict.get(student_id, "Unknown Student")
```

(iii) (I) Line 5: `for i in range(len(student_tuple[2])):`

(II) Line 9: `student_tuple = (student_tuple[0], "Alice Smith", student_tuple[2])`

(III) Line 11: `score_dict = {202501: student_tuple[2]}`

(IV) Line 12: `score_dict[202501].append(88)`

Marking Scheme:

(a) (i) 0.5 points for correct Subset: [20, 40, 60].
0.5 points for correct Popped: 70.
0.5 points for correct Final Data: [50, 60, 80].

(ii) (I) 0.5 points for identifying the correct line.

(II) 1 point for the explanation (Must mention "Lists are mutable" and "Keys must be immutable/hashable").

(III) 1 points for a valid code fix (Converting to tuple).
Note: The answer that switches a dictionary's keys and values will also be accepted as correct.

(b) (i) 0.5 points for each correct line. 2 points in total.

(ii) (I) 1 point for correct conditional judgment (`if score > 85`).
0.5 points for correct list comprehension structure and 0.5 points for correct multiplication by 1.1.

(II) 0.5 points for using the `get()` method.
0.5 points for correct default value ("Unknown Student").

(iii) (I) Removed "+1" to fix index out-of-bounds:
0.25 points for correct line number;
1 point for the correct code.

(II) Recreated tuple to fix the immutability error:
0.25 points for correct line number;
2 points for the correct code.

(III) Used int (hashable) instead of list as key:
0.25 point for correct line number;
1 point for the correct code.

(IV) Used int (hashable) instead of list as key:
0.25 point for correct line number;
1 point for the correct code.

Problem 4 [14 points] Functions

Solution:

(a) (i) (I) As follows:

Function call	Resulting value of `x`	Resulting value of `y`
`x, y = secret(101, -101)`	-101	101
`x, y = secret(0, 55)`	55	0
`x, y = secret(-73, -37)`	-37	-73
`x, y = secret(30624700, 30624770)`	30624770	30624700

(II) The purpose is to swap two integer variables.

(ii) No.

If at least one of the input parameters are zero, it will cause division by zero problem, which cannot perform the swapping correctly. This swapping also fails for integers with different signs.

(iii) No.

Since addition and subtraction of float variables would introduce precision errors, resulting in incorrect swapping results.

(b)
```python
def sum_even_nested(data: list) -> int:
    """
    Recursively calculates the sum of all even numbers in a nested list.
    """
    total = 0
    for element in data:
        if is_list(element):
            # If the element is a list, make a recursive call
            total += sum_even_nested(element)
        elif element % 2 == 0:
            # If the element is an integer, check if it's even
            total += element
    return total
```

Marking Scheme:

(a) (i) (I) 0.25 points for each correct value (2 points in total).

(II) 1 point for stating the correct purpose (swapping).

(ii)
- 1 point for answering no.
- 1 point for pointing out the division by zero problem.
- 1 point for stating the swapping also fails for integers with different signs

Notes:
- 0.5 points if the error is identified but the reason for the error is not explained.
- 0.5 points if the answer only identifies that there is problem with input values being 0 without pointing out the zero division error.
- 0.5 points if the response only mentions exceeding the maximum integer range.

(iii)
- 1 point for answering no.
- 1 point for pointing out the floating-point precision problem.

Notes:
- 0 points if only an example is given without explaining the issue.
- 0 points if the answer merely says the values "may not be equal" with no explanation.
- 0 points if the response claims the math library was not imported.
- -0.5 points for using "error" instead of "problem" when describing floating-point precision.
- -0.5 points if only addition or subtraction is mentioned without addressing floating point representation.
- -0.5 points if the response only says a float "becomes larger after some operations" without a precision rationale.
- -0.5 points for attributing the issue to rounding rather than floating point representation.
- -0.5 points for explanations that focus on "many decimal points" instead of the precision problem.

(b) (A) Recursive loops

1. Suggested solution
2. Recursive for-loop variants
3. "Unbox"/transform-then-reduce style

(B) No explicit loops

1. Pure recursion
2. Functional/comprehension-based recursion

For type (A): A.1 - A.3

- 0.5 points for initialization of accumulator
- 1 point for the loop
- 2 points for handling nested lists via recursion
- 2 points for handling even integers
- 0.5 points for correct final return

For type (B): B.1 - B.2

- 1.5 points for correct base case
- 2 points for correct recursive case for nested lists
- 2 points for correct handling of integers.
  -1 point for each line of error (e.g., did not handle odd integers under if statement, not using elif when needed, etc.)
- 0.5 points for correct final return

Others:

- 0 for non-recursive function and hard coding.

Problem 5 [7 points] Introduction to OOP

Solution:

(a)
```
#1 spent 5
W1: 5
W2: 160
W3: 5
```

(b) (i) `capacity` (Assigning the argument to the attribute)

(ii) `self.__level` (Checking the instance variable)

(iii) `my_phone.drain(500)` (Calling the method)

Marking Scheme:

(a) 1 point for each correct line of output. (4 points in total.)

(b) (i) 0.5 points for putting `capacity`.

(ii) 0.5 points for putting `self.__level`.

(iii) 2 points for putting `my_phone.drain(500)`.

Problem 6 [11 points] NumPy, Pandas, and Matplotlib

Solution:

(a)
```
import numpy as np
def process(img: np.ndarray) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    # Your part (i) code here

    center: np.ndarray = img[1:-1, 1:-1]

    # End of part (i) code

    # Your part (ii) code here

    rot90: np.ndarray = center[::-1, :].T

    # End of part (ii) code

    # Your part (iii) code here

    top_left: np.ndarray = rot90[:2, :2]
    bottom_right: np.ndarray = rot90[-2:, -2:]
    combined: np.ndarray = np.hstack([top_left, bottom_right])

    # End of part (iii) code
    return center, rot90, combined

img: np.ndarray = np.array([
    [50,  60,  70,  80,  90],
    [55,  65,  75,  85,  95],
    [60,  70,  80,  90, 100],
    [65,  75,  85,  95, 105],
    [70,  80,  90, 100, 110]
])

center, rot90, combined = process(img)
```

(b) (i)
```
daily_min_max: tuple[pd.Series, pd.Series] = \
    (temperature.min(axis=1), temperature.max(axis=1))
```

(ii)
```
ax.plot(daily_min_max[1], label='Daily Max')
ax.plot(daily_min_max[0], label='Daily Min')
```

Marking Scheme:

(a) (i) 0.5 point for correct slicing syntax, 0.5 point for correct indices.

(ii) 2 points for reversing rows, 1 point for transposing.

(iii) 1 point for correct extraction of top-left block, 1 point for correct extraction of bottom-right block, 1 point for correct concatenation. (3 points in total)

(b) (i) 0.75 points for using max() with axis=1, 0.75 points for using min() with axis=1, 0.5 points for creating the tuple. (2 points in total)
Note: 0.5 points is given for tuple creation if and only if min and max values are correctly created.

(ii) For each plot, 0.5 points for correctly specifying the data, 0.5 points for the correct label. (2 points in total)
Note:

- -0.5 points for any error in label assignment.
- -0.5 points for missing `label=...`.
- -0.25 points for calling wrong functions. E.g., `plt.ax.plot...`.
- 0.5 points only if `ax.set_label('...')`.

Problem 7 [13 points] Phrase Appearance Counting

Solution:

Line No.	Reasons and Corrections
3	Reason for the error: The filter function cannot make all characters into lower cases.
	Correction: `phrase_tokens: list[str] = list(map(lambda x: x.lower(), phrase.split(" ")))`
13	Reason for the error: The is operator cannot check whether the two lists have the same elements.
	Correction: `if window_lower == phrase_tokens:`
14	Reason for the error: The program does not increment the count by 1, but assigns the count to 1 every time.
	Correction: `appearance_dict[paragraph_number] += 1`
17	Reason for the error: The count for each paragraph is stored in the dictionary's value, but the program accesses the key.
	Correction: `total_count = sum(appearance_dict.values())`

Marking Scheme:

For each error:

- 0.5 points for identifying the line number.
- 1.0 points for stating the reasoning for the error.
- 1.5 points for the correction.

For the error in Line 13:

- 0.5 points for identifying the line number.
- 2 points for stating the reasoning for the error.
- 1.5 points for the correction.

13 points in total for 4 errors.

Problem 8 [20 points] COMP 1023 Administrative Work

Solution:

(a) (i)
```
def attendance(scores: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    count = np.count_nonzero(np.isnan(scores),axis=1)
    scores[np.isnan(scores)] = 0
    return scores.sum(axis=1), 10 - count
```

or

```
def attendance(scores: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    mask = np.isnan(scores)
    scores[mask] = 0
    att = scores.sum(axis=1)
    scores[~mask] = 1
    return att, scores.sum(axis=1)
```

(ii)
```
def exercise(scores: np.ndarray) -> np.ndarray:
    return np.sort(scores, axis=1)[:, 1:].sum(axis=1)
```

or

```
def exercise(scores: np.ndarray) -> np.ndarray:
    scores.sort(axis=1)
    return scores[:, 1:].sum(axis=1)
```

or

```
def exercise(scores: np.ndarray) -> np.ndarray:
    return scores.sum(axis=1) - scores.min(axis=1)
```

(iii)
```
def total(att: np.ndarray, count: np.ndarray, exe: np.ndarray) -> np.ndarray:
    return (att + exe) / (40 + count) * 10
```

(b) (i)
```
def pa_sum(submission: str) -> int:
    return sum(map(to_num, submission))
```

or

```
def pa_sum(submission: str) -> int:
    return sum(map(lambda c: to_num(c), submission))
```

(ii)
```
def pairwise(differences: np.ndarray) -> np.ndarray:
    return np.abs(differences[:, np.newaxis] - differences[np.newaxis, :])
```

or

```
def pairwise(differences: np.ndarray) -> np.ndarray:
    return np.abs(differences[np.newaxis, :] - differences[:, np.newaxis])
```

(iii)
```
def get_students(ids: np.ndarray, pairs: np.ndarray) -> tuple[int, int]:
    np.fill_diagonal(pairs, pairs.max() + 1)
    r, c = np.where(pairs.min() == pairs)
    return ids[r[0]], ids[c[0]]
```

or

```
def get_students(ids: np.ndarray, pairs: np.ndarray) -> tuple[int, int]:
    np.fill_diagonal(pairs, pairs.max() + 1)
    r, c = np.where(pairs.min() == pairs)[0]
    return ids[r], ids[c]
```

(Misc.) Example of the given function (not required, simply FYI):

```
def to_num(c: str) -> int:
    return ord(c)
```

Marking Scheme:

General notes:

- For all parts, **NO** points should be given if **ANY** loops or comprehension are used.
- For all parts in (a), **NO** points should be given if any **non-**NumPy functions are used.
- For each part, 0.5 points should be deducted if the return statement is missing.
- No points will be given for hard-coding.

(a) (i)
- 1 point for using `np.isnan` to get boolean array of whether values are `np.nan`.
- 1 point for getting number of `np.nan` using `np.count_nonzero`.
  - This can also be done via getting sum across axis 1 if `np.nan` values are set to 0.
- 1 point for removing `np.nan` values by changing them to 0. Otherwise, the sum will become `np.isnan`, as anything plus `np.isnan` leads to `np.isnan`.
- 0.5 points for getting the sum across columns.
- 0.5 points for getting `10 - count`.
  - This can also be done via setting `~np.nan` to 1 and other values 0, then getting the sum across axis 1.

(ii)
- 1 point for using `np.sort` or `np.ndarray.sort` on `scores`.
- 1 point for removing the smallest score via slicing.
- 1 point for getting the sum across columns.

or

- 1 point for getting the total sum across axis 1.
- 1 point for getting the minimum elements across axis 1.
- 1 point for subtracting the minimum value from the total sum.

(iii)
- 1 point for `att + exe`
- 1 point for the correct logic of adding `40` to `count`
- 1 point for the correct logic of calculation of total score.

General notes:

- For all parts, **NO** points should be given if **ANY** loops or comprehension are used.
- For parts (b)(i) and (b)(ii), **NO** points should be given if > 1 statements is used.
- For all parts in (b)(ii) and (b)(iii), **NO** points should be given if any **non-**NumPy functions are used.
- For each part, 0.5 points should be deducted if the return statement is missing.
- No points will be given for hard-coding.

(b) (i)
- 1 point for using the `map` function.
- 1 point for applying `to_num` to each character of `submission`.
- 1 point for getting the sum via `sum` function.

(ii)
- 1 point for expanding the dimensions of `differences` so that it can be broadcasted with itself for pairwise calculation.
- 1 point for subtraction between 2 broadcastable arrays.
- 1 point for using `np.abs` or `np.absolute`.

(iii)
- 1 point for realising the diagonal difference will always be 0, and should be removed using `np.fill_diagonal`.
- 1 point for filling it with `pairs.max() + k`, where `k` ≥ 1.
  - 0.5 points will be given for filling values with `pairs.max()` or equivalent.
- 1 point for correctly using `np.where` to get indices of the minimum difference.
  - 0.5 points will be given if `np.where` is used incorrectly.
- 0.5 points for getting the correct indices.
  - Using `np.argmin` correctly can get you the 1 point from `np.where` and 0.5 points for getting the correct indices, culminating in 1.5 points.
- 0.5 points for indexing `ids` with the acquired indices.

---------- END OF PAPER ----------