COMP 1023 Midterm Exam - Fall 2025 - HKUST

Date: October 25, 2025 (Saturday)
Time Allowed: 2 hours, 1:00–3:00 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **6** questions on **19** pages (including this cover page, honor code, consent page and 4 blank pages at the end).
3. Write your answers in the space provided in black/blue ink. *NO pencils please, otherwise you are not allowed to appeal for any grading disagreements.*
4. All programming codes in your answers must be written in the Python version as taught in the 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		Seat Number	

For T.A. Use Only

Problem	Topic	Score
1	True/False Questions	/ 10
2	Branching and Looping Statements	( ) + ( ) = / 10
3	Functions and Lists	( ) + ( ) + ( ) + ( ) = / 18
4	Mini Store System	( ) + ( ) + ( ) + ( ) = / 20
5	Number Guessing Game	( ) + ( ) + ( ) = / 26
6	Sales of Car Accessories	( ) + ( ) + ( ) = / 16
	Total	/ 100

The HKUST Academic Honor Code

Honesty and integrity are central to the academic work of HKUST. Students of the University must observe and uphold the highest standards of academic integrity and honesty in all the work they do throughout their program of study.

As members of the University community, students have the responsibility to help maintain the academic reputation of HKUST in its academic endeavors.

Sanctions will be imposed if students are found to have violated the regulations governing academic integrity and honesty.

Declaration of Academic Integrity

I confirm that I have answered the questions using only materials specifically approved for use in this examination, that all the answers are my own work, and that I have not received any assistance during the examination.

Signature: _____

You are required to sign this honor code.

Consent for Use of Exam Scripts in AI Training

We are seeking your consent to use your exam scripts for the purpose of training an artificial intelligence system that will enhance our online and onsite computer-based examination platform. This AI will help improve the quality and efficiency of our assessments. Please note that your identity, including your name and student ID, **will not** be used in the training process.

Your contributions are invaluable, and we want to ensure that you are comfortable with this process. If you agree to allow us to use your exam scripts for this purpose, please sign to indicate your consent. Thank you for your support in helping us create a better examination experience for everyone.

Declaration of Consent

I confirm that I agree to allow my exam scripts to be used for training the AI.

Signature: _____

You are not required to sign if you do not agree to allow your exam scripts to be used.

Problem 1 [10 points] True/False Questions

Solution:

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

Marking Scheme:

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

Problem 2 [10 points] Branching and Looping Statements

Solution:

(a)

Element	1	None	3	1	-7	-4	5
Value of `total` after processing the element	0	2	6	5	12	8	/

(b) 162, run out of items naturally

Marking Scheme:

(a) 1 point for each correct total value (7 points in total).

(b) 1.5 points for the correct return value and 1.5 points for the correct reason for termination (3 points in total).

Problem 3 [18 points] Functions and Lists

Solution:

(a) (i)
```python
def is_divisible(n: int, k: int) -> bool:
    return n % k == 0
```

(ii)
```python
def is_prime(n: int) -> bool:
    if n == 2:
        return True
    for i in range(2, int(n**0.5) + 1):
        if is_divisible(n, i):
            return False
    return True
```

(iii)
```python
def get_primes(n: int) -> tuple[int, list[int]]:
    count: int = 0
    primes: list[int] = []

    for i in range(2, n + 1):
        if is_prime(i):
            count += 1
            primes.append(i)

    return count, primes
```

(b)
```python
def rotate_clockwise(lst: list[list[int]]) -> list[list[int]]:
    n: int = len(lst)
    # Create a new n x n 2D list filled with zeros
    rotated: list[list[int]] = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            # The element at [i][j] goes to [j][n-1-i] in clockwise rotation
            rotated[j][n - 1 - i] = lst[i][j]
    return rotated
```

Marking Scheme:

(a) 1 point for problem (i), 4 points for problem (ii), and 6 points for problem (iii), for a total of 11 points.

(i) 1 point:

- 0.5 points for checking divisibility using the modulus operator (%) without loops.
- 0.5 points for correctly returning a boolean value.

(ii) 4 points:

- 1 point for using the `is_divisible` function **correctly** from part (a)(i).
- 1 point for breaking the loop early when the divisor is found.
- 1 point for running the loop from 2 to at least $\sqrt{n}$ (it is acceptable if the loop runs from 2 to n).
- 1 point for correctly handling the edge case (i.e., `n==2`).
  - If the loop iterates from 2 to `n`, it is acceptable not to include explicit edge-case handling.
  - If the loop starts from 0 (i.e., `for` divisor in `n`), the edge cases `divisor == 0`, `divisor == 1` should be handled.
  - If the loop runs beyond `n`, the edge case `divisor == n` should be handled.

(iii) 6 points:

- 1.5 points for using the `is_prime` function from part (a)(ii).
- 1.5 points for running the loop from 2 to n+1.
- 1.5 points for correctly counting the primes.
- 1.5 points for correctly returning the counted value.

(b) 7 points:

- 0.5 points for `n = len(lst)`.
- 2 points for initializing a new n×n 2D list.
- 1.5 points for having two loops that run from 0 to n.
- 2 points for the value assignment.
- 1 point for correctly returning the rotated 2D list. (No point with only the return statement without the correct variable.)
- Deduct 0.25 points for each syntax error.
- 0 point for hardcoding.
- Additional functions not allowed.

Other common solutions:

```
def rotate_clockwise(lst):
    n = len(lst)
    rotated = []
    for i in range(n):
        new_row = []
        for j in range(n-1, -1, -1):  # -0.25 for each mistake in the range function
            new_row.append(lst[j][i])
        rotated.append(new_row)
    return rotated

def rotate_clockwise(lst):
    n = len(lst)
    rotated = []
    for i in range(n):
        new_row = []
        for j in range(n):
            new_row.append(lst[n-j-1][i])
        rotated.append(new_row)
    return (rotated)
```

For these solutions, the marking scheme is different in the following way:

- Initialization of `rotated` incorporates several lines to get full 2 points.
- 1.5 points for having two loops.
- 2 points for correct indexing and appending.

Common mistakes:

- Treating `lst` as `1st`, and naming other variables with `2nd`, etc. are incorrect since Python does not allow variable names to start with a digit.
- Incorrect use of `reverse()` and `reversed()`.

Problem 4 [20 points] Mini Store System

Solution:

(a)
```
def print_balance(balance: int) -> None:
    print("Your balance is $" + str(balance))
```

(b)
```
def print_bag(bag: list[str]) -> None:
    print("Your bag contains: ", end="")
    for item in bag:
        print(item, end=" ")
    print()
```

Another version:

```
def print_bag(bag: list[str]) -> None:
    print("Your bag contains:", *bag)
```

(c)
```
def put_in_bag(bag: list[str], item: str, amount: int) -> list[str]:
    for _ in range(amount):
        bag.append(item)
    return bag
```

(d)
```
def purchase_item(balance: int, item: str, price: int,
                  amount: int = 1) -> tuple[int, str, int]:
    if balance >= price * amount:
        balance -= price * amount
        return balance, item, amount
    else:
        return balance, item, 0
```

Another version:

```
def purchase_item(balance: int, item: str, price: int, amount: int = 1) \
    -> tuple[int, str, int]:
    return (balance - price * amount, item, amount) \
            if balance >= price * amount else (balance, item, 0)
```

Marking Scheme:

(a) **Print the Balance (2 points)**

- 0.5 points for correctly defining the function with the name `print_balance`.
- 0.5 points for correctly defining the function parameter `balance`.
- 0.5 points for correctly printing "Your balance is $".
- 0.5 points for correctly printing `balance`.

(b) **Print the Bag (6 points)**

- 0.5 points for correctly defining the function with the name `print_bag`.
- 0.5 points for correctly defining the function parameter `bag`.
- 1 point for correctly printing `Your bag contains:`.
- 2 points for correctly using a loop to print items.
- 2 points for correctly printing the statement and output format.

(c) **Put Items in the Bag (4 points)**

- 0.5 points for correctly defining the function with the name `put_in_bag`.
- 1 point for correctly defining the function parameters `bag, item, amount`.
- 2 points for correctly using a loop to add multiple items.
- 0.5 points for correctly returning the updated bag.

(d) **Purchase an Item (8 points)**

- 0.5 points for correctly defining the function with the name `purchase_item`.
- 1.5 points for correctly defining the function parameters `balance, item, price, amount`.
- 1 point for correctly handling the default parameter value, i.e., `amount = 1`.
- 2.5 points for correctly implementing the balance deduction logic.
- 2.5 points for correctly returning the value (tuple structure).

Problem 5 [26 points] Number Guessing Game

Solutions:

(a)
```python
def initialize_game() -> list[int | bool]:
    secret_number: int = random.randint(1, 100)
    return [secret_number, 0, 10, 1, 100, False]
```

(b)
```python
def get_valid_guess(game_state: list[int | bool]) -> int:
    while True:
        guess_input: str = input(f"Attempt {game_state[1] + 1} \
                                 (Range: {game_state[3]}-{game_state[4]}): ")

        if not guess_input.isdigit():
            print("Invalid input! Please enter a whole number.")
            continue

        guess: int = int(guess_input)

        if guess < game_state[3] or guess > game_state[4]:
            print(f"Please enter a number between {game_state[3]} and {game_state[4]}.")
            continue

        return guess
```

(c)
```python
def process_guess(guess: int, game_state: list[int | bool]) -> list[int | bool]:
    game_state[1] += 1

    if guess == game_state[0]:  # secret_number
        print(f"Congratulations! You guessed the number {game_state[0]} in\
              {game_state[1]} attempts!")
        game_state[5] = True    # game_over
    elif guess > game_state[0]:   # secret_number
        print("Too high!")
        game_state[4] = guess - 1  # high_bound
    else:
        print("Too low!")
        game_state[3] = guess + 1  # low_bound

    print()

    return game_state
```

Marking Scheme:

(a) **Initialize the Game (4 points)**

- 2 points for correctly generating a random number in the range of 1 to 100 (inclusive). Deduct 1 point for each incorrect boundary (upper or lower) if a number is generated.
- 2 points for correctly returning the list containing the game state. Deduct 0.5 points for each incorrect variable with maximum deduction of 2 points. Deduct 0.5 points for no return or return the list in an incorrect order.
- Deduct 0.25 points for each type of syntax error.

(b) **Get Valid Guess (11 points)**

- 1 point for correctly obtaining the user's input guess number.
- 1.5 points for correctly showing the prompt text.
- 3 points for correctly checking for non-digit input, displaying the error message `Invalid input! Please enter a whole number`, and allowing the user to enter the guess number again.
- 1.5 points for correctly converting the guess number from a string to an integer.
- 3 points for correctly checking if the guess number is within the legal range, displaying the error message `Please enter a number between ... and ...`, and allowing the user to enter the guess number again.
- 1 point for correctly returning the guess number.

(c) **Process Guess (11 points)**

- 1 point for correctly increasing the number of attempts by 1.
- 1 point for correctly performing the equality test for the secret number.
- 1.5 points for correctly showing the message `Congratulations! You guessed the number ... in ... attempts.`
  - 0.5 points for the correct string.
  - 0.5 points for each of the two variables to fit into the f-string.
  - Common deductions were for incorrect variables or missing the word "attempts".
- 1 point for correctly setting the `game_over` state to `True`.
- 1 point for correctly performing the "greater than" test for the secret number.
- 1 point for correctly showing the message `Too high!`.
- 1 point for correctly setting the high bound to guess - 1.
- 1 point for correctly performing the "less than" test for the secret number.
- 1 point for correctly showing the message `Too low!`.
- 1 point for correctly setting the low bound to guess + 1.
- 0.5 points for correctly returning the list of game state.

Note:

- 0 points for using undefined or fabricated variables (e.g., `secret_number`, or using int/list as variable names).
- Several students use "`secret_number`" without defining it. We give 0 points for each line involving fake variables.
- We see several students use "`int`" and "`list`" as variable names, this also received 0 points for the lines involving.

Problem 6 [16 points] Sales of Car Accessories

Solution:

(a)
```
def transpose(sales_list: list[list[int]]) -> list[list[int]]:
    return [
        [row[i] for row in sales_list]
        for i in range(len(sales_list[0]))
    ]
```

or

```
def transpose(sales_list: list[list[int]]) -> list[list[int]]:
    return [
        [sales_list[j][i] for j in range(len(sales_list))]
        for i in range(len(sales_list[0]))
    ]
```

(b)
```
def shift(sales_list: list[list[int]], days: int) -> list[list[int]]:
    return [
        row[days % len(row):] + row[:days % len(row)]
        for row in sales_list
    ]
```

(c)
```
def shift_right(sales_list: list[list[int]], days: int) -> list[list[int]]:
    return shift(sales_list, -(days % len(sales_list[0])))
```

or

```
def shift_right(sales_list: list[list[int]], days: int) -> list[list[int]]:
    return shift(sales_list, len(sales_list[0]) - (days % len(sales_list[0])))
```

Marking Scheme:

For all parts:

- Deduct 2 points if the list is modified in-place.
- Score is capped at 4 for part (a), 3 for parts (b) and (c) if more than one line is used.
- Score is capped at 2 if > 2 looping statements (excluding comprehension loops) are used.
- Deduct 0.5 points if `return` statement is missing.
- **0 points if lambda functions or semi-colons are used.**

(a)
- 1 point for getting the number of columns via `len(sales_list[0])`.
- 1 points for correctly iterating through the columns of `sales_list` in the outer comprehension, e.g.: `for i in range(len(sales_list[0]))`.
- 1 point for getting the number of rows via `len(sales_list)`.
  - This point is automatically given if the student iterates through the rows directly.
- 1 point for correctly iterating through `sales_list` in the inner comprehension.
- 2 points for correctly getting the column values via `row[i]` or `sales_list[j][i]`.

(b)
- 1 point for realizing the value of `days` could be larger than `len(row)`.
- 1 point for using `%` to make the used value be in the range of [`1`, `len(row)`). **(This would handle negative `days` as well)**
  - No points given if a literal (e.g., `7`) is used instead of `len(row)`
- 0.75 points for getting the elements from indices `days` to `len(row)` via slicing.
- 0.75 points for getting the elements from indices `0` to `days` via slicing.
- 1 point for getting the number of columns via `len(sales_list)`.
  - This point is automatically given if the student iterates through the rows directly.
- 0.5 points for correctly iterating through `sales_list` via list comprehension.

(c)
- 1 point if `sales_list` is passed into `shift()`
- 2 points if `shift` is a tail call, i.e. the statement should be `return shift(...)`
- 1 point for realizing the value of `days` could be larger than `len(row)`.
  - This point is automatically given if it is properly handled in part (b) already.
- If the student uses negation:
  - 1 point given **only if student's answer in part (b) handled it**.
- Otherwise:
  - 1 point for deducting the value from `len(sales_list[0])` as the second parameter.

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

/* Rough work */

/* Rough work */

/* Rough work */

/* Rough work */