COMP 1023 Final Exam - Fall 2025 - HKUST
Question Book

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 **27** 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			
Student ID			
Venue	Sports Hall / LG4 / Rm 2407 (Circle the venue)	Seat Number	

This page is intentionally left blank.

Problem 1 [10 points] True/False Questions

Indicate whether the following statements are true or false by putting **T** or **F** in the given table in the answer book. You get 1 point for each correct answer.

(a) The two print statements in the following program would print the exact same thing.

```
def main() -> None:
    number: int = 10
    string: str = "10"
    print(number + int(string))
    print(str(number) + string)

if __name__ == "__main__":
    main()
```

(b) In Python, loop statements cannot have an else clause associated with them.

(c) In Python, the `*args` and `**kwargs` syntax in function definitions allows you to pass a variable number of positional and keyword arguments, respectively.

(d) The following program runs without any errors.

```
def func(a: int, b: int, c: int) -> None:
    pass

def main() -> None:
    func(a=1, 2, 3)

if __name__ == "__main__":
    main()
```

(e) The expression `["Python", "is", "fun!"][:4]` would lead to an error.

(f) The following Python program defines a function named `add_item` that modifies a list. Because mutable objects like lists are passed to functions by object reference, the list `data` is modified by the function call, and its final value is `[10, 20, 100]`.

```
def add_item(my_list: list[int]) -> None:
    my_list.append(100)

def main() -> None:
    data: list[int] = [10, 20]
    add_item(data)

if __name__ == "__main__":
    main()
```

(g) The output of the following program is `[1, 3]`.

```
def append_to(element: int, target: list[int] = []) -> list[int]:
    target.append(element)
    return target

def main() -> None:
    list1 = append_to(1)
    list2 = append_to(2, [])
    list3 = append_to(3)

    print(list1)

if __name__ == "__main__":
    main()
```

(h) In a Python dictionary, you can use any data type (e.g., integers, strings, lists, tuples) as a key.

(i) In NumPy, assigning an existing array to a new variable (e.g., `b = a`) creates a deep copy of the data. Therefore, modifying `b` will not affect the original array `a`.

(j) The following program runs without any errors and prints `6`.

```
class Q:
    def __init__(self, x:list[int] = [1]):
        self.val = x
    def func(self) -> None:
        self.val.append(1)
    def question(self, num: int) -> list[int]:
        for _ in range(num):
            self.func()
        return sum(self.val)

def main() -> None:
    q = Q
    print(q.question(5))

if __name__ == "__main__":
    main()
```

Problem 2 [10 points] Branching and Looping

A teacher needs to process 4 students' after-class assignment scores (integers out of 100, scores: [85, 58, 72, 60]). The task uses looping to traverse scores and branching to judge "Qualified" (score ≥ 60) or "Unqualified" (score < 60). Answer the questions below.

(a) [2 points] The code below uses a for loop to iterate over the 4 students' scores and a conditional expression to show qualification results. Write the exact output (include all lines, no extra/missing content).

```
assignment_scores: list[int] = [85, 58, 72, 60]
for score in assignment_scores:
    print(f"Score: {score}", 'Qualified' if score >= 60 else 'Unqualified')
```

(b) [1 point] The code below counts how many students are Unqualified (score < 60) using a for loop. Fill in the missing blank to make the code output number of unqualified students from the list [85, 58, 72, 60].

```
assignment_scores: list[int] = [85, 58, 72, 60]
unqualified_count: int = 0  # Initialize counter for unqualified students

for score in assignment_scores:
    if _____:  # Check if the score is unqualified
        unqualified_count += 1

print(f"Number of unqualified students: {unqualified_count}")
```

(c) [4 points] Write a Python code using a **while loop** to calculate the total sum of the 4 students' assignment scores ([85, 58, 72, 60]). The code must include:

- Initialization of variables (total sum and loop control variable).
- The while loop (correct condition and update of the loop control variable).
- A print statement to output the calculated total sum as follows:

  `Total assignment score: 275`

(d) [3 points] Write a Python code using a **for loop** to find the highest score among the 4 students' assignment scores ([85, 58, 72, 60]). The code must use an **if statement** to compare scores and include a print statement to output the highest score found as follows:

`Highest assignment score: 85`

Remark: You are not allowed to use any functions except for `print()`.

Problem 3 [15 points] Collections

(a) (i) [1.5 points] Analyze the following Python code involving list slicing and methods. Write down the exact output.

```
def main():
    data: list[int] = [10, 20, 30, 40, 50, 60, 70, 80]

    # Operation 1: Slicing with step
    subset: list[int] = data[1:7:2]
    print(f"Subset: {subset}")

    # Operation 2: Negative indexing and popping
    val: int = data.pop(-2)
    print(f"Popped: {val}")
    print(f"Final Data: {data[-3:]}")

if __name__ == "__main__":
    main()
```

(ii) The following code snippet attempts to use a dictionary to store coordinates. However, it crashes with a runtime error.

```
1 locations = {}
2 home_coords = [22, 114]
3
4 locations[home_coords] = "Home"
5
6 print(locations)
```

(I) [0.5 points] Write the line number where the error occurs.

(II) [1 point] Explain why this error occurs.

(III) [1 point] What can you do to fix the error while preserving the logic of the code? Explain briefly.

(b) Suppose we are developing a simple student score management system. The system involves storing basic student information, exam scores, and statistical analysis. Complete the following tasks based on the given code framework and requirements.

(i) [2 points] Write the exact output of the following code.

```
def manage_student_scores() -> None:
    score_list: list[float] = [92.5, 88.0, 76.5, 95.0, 82.0]

    info_tuple: tuple[str, int, list[float], str] = \
      ("Alice", 202501, score_list, "Math")
    student_dict: dict[int, str] = \
      {202501: "Alice", 202502: "Bob", 202503: "Charlie"}

    print(score_list[1:4:2])

    info_tuple[2][0] = 90.0
    print(info_tuple[2][0])

    print(202504 in student_dict)

    new_tuple: tuple[str, int, str, float] = \
      info_tuple[:2] + ("Physics", 89.0)  # Concatenate two tuples
    print(new_tuple[-1])

manage_student_scores()
```

(ii) The `process_data` function is used to filter and adjust student scores, and retrieve student names. Complete the two blanks (`blank1` and `blank2`) with one line of code each to perform the following tasks:

(I) [2 points] `blank1`: **Use a list comprehension** to filter `scores > 85` from `score_list`, then multiply each filtered score by 1.1 (extra credit coefficient).

(II) [1 point] `blank2`: **Use a dictionary method** to get the student's name from `student_dict` via `student_id`. If `student_id` is not in the dictionary, return the default value `"Unknown Student"`.

```
def process_data(score_list: list[float], student_id: int,
                 student_dict: dict[int, str]) -> tuple[list[float], str]:
    # blank1: Filter and adjust scores (list comprehension)
    adjusted_scores = _____blank1_____
    # blank2: Get student name with default value (dictionary method)
    student_name = _____blank2_____
    return adjusted_scores, student_name

scores: list[float] = [80, 92, 78, 88, 95]
stu_dict: dict[int, str] = {202501: "Alice", 202502: "Bob"}
result_scores, name = process_data(scores, 202504, stu_dict)
print(result_scores)  # Expected output: [101.2, 96.8, 104.5]
print(name)           # Expected output: Unknown Student
```

(iii) [6 points] The code below is used to count passing scores (≥ 60), update student names, and store scores in a dictionary. There are 4 errors in the code. Fill in the line number of each error, and write down the correct version of the line of code in the answer book.

```
1  def count_scores():
2      # Student data: (student_id, name, [scores]) → A tuple contains a mutable list
3      student_tuple = (202501, "Alice", [58, 90, 75])
4      pass_count = 0
5      for i in range(len(student_tuple[2]) + 1):
6          if student_tuple[2][i] >= 60:
7              pass_count += 1
8      # Update Alice's name to her full name
9      student_tuple[1] = "Alice Smith"
10     # Try to use a list as a dictionary key
11     score_dict = {[202501]: student_tuple[2]}
12     score_dict[[202501]].append(88)
13
14     print("pass_count:", pass_count)
15     print("student_tuple:", student_tuple)
16     print("score_dict:", score_dict)
17
18 count_scores()
```

After making corrections, the program should output the following when executed:

```
pass_count: 2
student_tuple: (202501, 'Alice Smith', [58, 90, 75, 88])
score_dict: {202501: [58, 90, 75, 88]}
```

Problem 4 [14 points] Functions

(a) In this question, you are given 3 secret functions. Your task is to study these secret functions and answer the following questions.

(i) Consider the following function.

```
def secret(x: int, y: int) -> tuple[int, int]:
    x = x + y
    y = x - y
    x = x - y
    return x, y
```

(I) [2 points] What would be the resulting values of `x` and `y` from the following function calls?

- `x, y = secret(101, -101)`
- `x, y = secret(0, 55)`
- `x, y = secret(-73, -37)`
- `x, y = secret(30624700, 30624770)`

(II) [1 point] Based on all the examples above, identify the purpose of the `secret` function.

(ii) [3 points] Consider the following function. The difference from the `secret` function in part (i) is the operators involved.

```
def secret2(x: int, y: int) -> tuple[int, int]:
    x = x * y
    y = x // y
    x = x // y
    return x, y
```

Can the `secret2` function achieve the same purpose as the `secret` function in part (i), for any possible integer input parameters? Why?

Hint: Consider some of the function calls in part (a)(i).

(iii) [2 points] Consider the following function, which extends `secret()` in part (i) to floats. The difference from the `secret` function in part (i) is the data type of the input parameters.

```
def secret3(x: float, y: float) -> tuple[float, float]:
    x = x + y
    y = x - y
    x = x - y
    return x, y
```

Can the `secret3` function achieve the same purpose as the `secret` function in part (i), for any possible float input parameters? Why?

Hint: Consider a classic example that `0.1 + 0.2` in Python results in `0.30000000000000004`.

(b) [6 points] Write a **recursive** function named `sum_even_nested(data)`. This function takes a nested list `data` as an argument, which may contain integers and other lists. The function should compute and return the sum of all even numbers within this nested structure. You must use recursion to process the nested lists.

For example:

- `sum_even_nested([1, 2, [3, 4, [5]], 6])` should return `12` (because 2 + 4 + 6 = 12).
- `sum_even_nested([10, [20, 30], [[40], 50]])` should return `150` (because 10 + 20 + 30 + 40 + 50 = 150).
- `sum_even_nested([1, 3, [5, 7]])` should return `0`.

Assume the function `is_list`, which checks if an element is a list, is defined for you to use in your implementation.

```
def is_list(element):
    return type(element) == list
```

The following shows a main function that uses the function `sum_even_nested(data)` with different inputs. It also shows what the correct outputs should be.

```
# Your implementation of sum_even_nested(data) is here

def main():
    list1 = [1, 2, [3, 4, [5]], 6]
    result1 = sum_even_nested(list1)
    print(f"The sum of even numbers in {list1} is: {result1}")

    list2 = [10, [20, 30], [[40], 50]]
    result2 = sum_even_nested(list2)
    print(f"The sum of even numbers in {list2} is: {result2}")

    list3 = [1, 3, [5, 7]]
    result3 = sum_even_nested(list3)
    print(f"The sum of even numbers in {list3} is: {result3}")

    list4 = []
    result4 = sum_even_nested(list4)
    print(f"The sum of even numbers in {list4} is: {result4}")

if __name__ == "__main__":
    main()
```

Output:

```
The sum of even numbers in [1, 2, [3, 4, [5]], 6] is: 12
The sum of even numbers in [10, [20, 30], [[40], 50]] is: 150
The sum of even numbers in [1, 3, [5, 7]] is: 0
The sum of even numbers in [] is: 0
```

Problem 5 [7 points] Introduction to OOP

(a) [4 points] Analyze the execution of the following Python code. Pay close attention to **variable assignment** and **object references**. Write the exact output.

```
class EWallet:
    def __init__(self, id_num: int, cash: float) -> None:
        self.id = id_num
        self.cash = cash

    def spend(self, amount: float) -> None:
        if self.cash >= amount:
            self.cash -= amount
            print(f"#{self.id} spent {amount}")
        else:
            print(f"#{self.id} rejected")

    def transfer(self, target: 'EWallet', amount: float) -> None:
        if self.cash >= amount:
            self.cash -= amount
            target.deposit(amount)

    def deposit(self, amount: float) -> None:
        self.cash += amount

# Main Execution
w1: EWallet = EWallet(1, 100)
w2: EWallet = EWallet(2, 50)
w3: EWallet = w1

w2.deposit(20)
w3.transfer(w2, 90)
w1.spend(5)

print(f"W1: {w1.cash}")
print(f"W2: {w2.cash}")
print(f"W3: {w3.cash}")
```

(b) [3 points] Implement the logic for the `Battery` class below, which manages the processes of charging and draining power.

- `__init__`: Sets up the battery to be fully charged initially.
- `drain`: Reduces battery level only if there is enough power.

Fill in the blanks labeled Part (i), Part (ii), and Part (iii) so that the program produces the expected output.

```
class Battery:
    def __init__(self, capacity: int) -> None:
        self.__capacity: int = capacity
        # Initialize the current level to be equal to the capacity
        self.__level: int = _____ Part (i) _____

    def drain(self, amount: int) -> bool:
        if _____ Part (ii) _____ >= amount:
            self.__level -= amount
            return True
        return False

    def get_status(self) -> str:
        return f"{self.__level}/{self.__capacity} mAh"

def main() -> None:
    my_phone: Battery = Battery(3000) # 3000 mAh battery

    if _____ Part (iii) _____:
        print("Using App...")
    else:
        print("Low Battery!")

    print(my_phone.get_status())

if __name__ == "__main__":
    main()
```

Expected output:

```
Using App...
2500/3000 mAh
```

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

(a) Below is a piece of Python code with three missing parts. This code processes a **5×5** grayscale image stored as a NumPy array. Complete the missing parts based on the provided instructions.

Remark: You are NOT allowed to use any loops or comprehensions in your answers.

```
import numpy as np

def process(img: np.ndarray) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    # (i) [1 point] First, use array slicing to extract the center 3x3 region
    # from img, i.e.,
    # np.array([ [65, 75, 85],
    #            [70, 80, 90],
    #            [75, 85, 95] ])
    # Assign this region to a variable named "center".
    # Your part (i) code here

    # (ii) [3 points] Then, rotate "center" by 90 degrees clockwise, i.e.,
    # np.array([ [75, 70, 65],
    #            [85, 80, 75],
    #            [95, 90, 85] ])
    # Assign this rotated region to a variable named "rot90".
    # You are NOT allowed to use np.rot90().
    # Your part (ii) code here

    # (iii) [3 points] Finally, extract the top-left 2x2 and bottom-right
    # 2x2 blocks from "rot90" in part (a)(ii), then concatenate
    # them horizontally into a 2x4 array, i.e.,
    # np.array([ [75, 70, 80, 75],
    #            [85, 80, 90, 85] ])
    # Assign this 2x4 array to a variable named "combined".
    # Your part (iii) code here

    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) [4 points] The following table shows the average hourly temperature (in °C) recorded over a week.

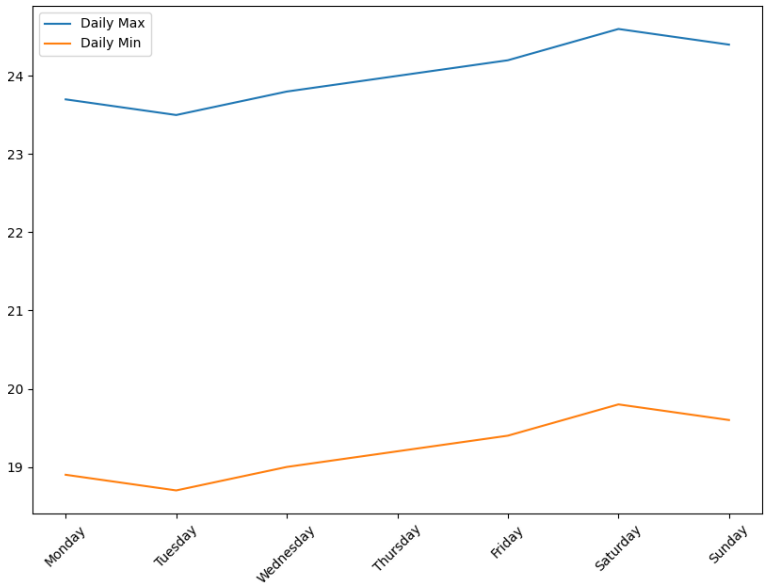
Time	00:00	02:00	04:00	06:00	08:00	10:00	12:00	14:00	16:00	18:00	20:00	22:00
Monday	20.1	19.4	18.9	19.0	20.2	21.8	23.1	23.7	23.3	22.4	21.4	20.5
Tuesday	19.8	19.2	18.7	18.8	20.0	21.6	22.9	23.5	23.1	22.2	21.2	20.3
Wednesday	20.0	19.5	19.0	19.1	20.3	21.9	23.2	23.8	23.4	22.5	21.5	20.6
Thursday	20.3	19.7	19.2	19.3	20.5	22.1	23.4	24.0	23.6	22.7	21.7	20.8
Friday	20.5	19.9	19.4	19.5	20.7	22.3	23.6	24.2	23.8	22.9	21.9	21.0
Saturday	21.0	20.3	19.8	19.9	21.1	22.7	24.0	24.6	24.2	23.3	22.3	21.4
Sunday	20.7	20.1	19.6	19.7	20.9	22.5	23.8	24.4	24.0	23.1	22.1	21.2

This data is stored in a CSV file named `temperature.csv`, as shown below, which consists of the table data.

```
Time,00:00,02:00,04:00,06:00,08:00,10:00,12:00,14:00,16:00,18:00,20:00,22:00
Monday,20.1,19.4,18.9,19.0,20.2,21.8,23.1,23.7,23.3,22.4,21.4,20.5
Tuesday,19.8,19.2,18.7,18.8,20.0,21.6,22.9,23.5,23.1,22.2,21.2,20.3
Wednesday,20.0,19.5,19.0,19.1,20.3,21.9,23.2,23.8,23.4,22.5,21.5,20.6
Thursday,20.3,19.7,19.2,19.3,20.5,22.1,23.4,24.0,23.6,22.7,21.7,20.8
Friday,20.5,19.9,19.4,19.5,20.7,22.3,23.6,24.2,23.8,22.9,21.9,21.0
Saturday,21.0,20.3,19.8,19.9,21.1,22.7,24.0,24.6,24.2,23.3,22.3,21.4
Sunday,20.7,20.1,19.6,19.7,20.9,22.5,23.8,24.4,24.0,23.1,22.1,21.2
```

Using Pandas and Matplotlib to perform the following tasks:

- Compute the daily minimum and maximum temperatures, and provide the answer as a tuple of two Pandas Series named `daily_min_max`, ensuring that the minimum is the first value and the maximum is the second value.
- Create a line plot that displays the daily maximum and minimum temperatures over a 7-day period as shown below.



The following is a provided code snippet for the tasks, but it contains some missing lines. Your job is to fill in the missing parts of the code to successfully compute the daily minimum and maximum temperatures and generate the line plot.

```
import pandas as pd
import matplotlib.pyplot as plt

# To analyze this data, we use the Pandas library in Python to read the
# CSV file and store the information in a DataFrame.
temperature: pd.DataFrame = pd.read_csv('temperature.csv', index_col=0)

# Create the tuple here

fig, ax = plt.subplots(figsize=(10, 7))

# Add a statement below to plot the daily maximum temperature data, and
# set the label for this line as "Daily Max".

# Add a statement below to plot the daily minimum temperature data, and
# set the label for this line as "Daily Min".

ax.legend()
plt.xticks(rotation=45)
fig.savefig("temperature.png")
```

Problem 7 [13 points] Phrase Appearance Counting

"Let It Be" is a well-known song that conveys a message of comfort and reassurance. Your task is to count how many times the phrase "let it be" (case insensitive) appears in the lyrics of the song using a Python program.

After putting in some effort at midnight, you implemented a program to perform the given task. However, after you wake up, you realize that the program has several bugs, and you need to begin the debugging journey.

There are 5 errors in the `count_phrase_appearance` function (i.e., line 1 - 18), and you need to find and fix all of them. For each error, please indicate the line number, explain the reason for the error, and correct that line using a one-line statement. You are NOT allowed to insert additional lines, remove some lines, or modify the main function.

```
1  def count_phrase_appearance(article: str, phrase: str) -> None:
2      appearance_dict: dict[int, int] = {}
3      phrase_tokens: list[str] = list(filter(lambda x: x.lower(), phrase.split(" ")))
4      article_by_paragraphs: list[str] = article.split("\n\n")
5      paragraph_number: int = 0
6      for paragraph in article_by_paragraphs:
7          paragraph_number += 1
8          appearance_dict[paragraph_number] = 0
9          text_tokens: list[str] = paragraph.split(" ")
10         for i in range(0, len(text_tokens)):
11             window: list[str] = [text for text in text_tokens[i:i+len(phrase_tokens)]]
12             window_lower: list[str] = list(map(lambda x: x.lower(), window))
13             if window_lower is phrase_tokens:
14                 appearance_dict[paragraph_number] = 1
15     for key, value in appearance_dict.items():
16         print(f"Paragraph {key}: the phrase appears {value} time(s).")
17     total_count = sum(appearance_dict.keys())
18     print(f"In total, the phrase appears {total_count} time(s).")
19
20 def main() -> None:
21     # The following statements are used to read the entire content
22     # (with all the newline characters) of "let-it-be.txt".
23     input_file = open("let-it-be.txt", 'r')
24     article: str = input_file.read()
25     input_file.close()
26     search_phrase: str = "Let it be"
27     count_phrase_appearance(article, search_phrase)
28
29 if __name__ == "__main__":
30     main()
```

With all the corrections done, the program should have the following output.

```
Paragraph 1: the phrase appears 2 time(s).
Paragraph 2: the phrase appears 5 time(s).
Paragraph 3: the phrase appears 2 time(s).
Paragraph 4: the phrase appears 5 time(s).
Paragraph 5: the phrase appears 5 time(s).
Paragraph 6: the phrase appears 5 time(s).
Paragraph 7: the phrase appears 2 time(s).
Paragraph 8: the phrase appears 10 time(s).
In total, the phrase appears 36 time(s).
```

You have been provided the following text file named `let-it-be.txt`, which contains the complete lyrics of the song. This file is used in the provided program. Each paragraph is separated by two newline characters. For simplicity, all punctuation has been removed in advance. The phrase "Let it be" (case insensitive) is highlighted for your reference.

When I find myself in times of trouble Mother Mary comes to me Speaking words of wisdom **let it be** And in my hour of darkness she is standing right in front of me Speaking words of wisdom **let it be**

Let it be let it be let it be let it be Whisper words of wisdom **let it be**

And when the broken hearted people living in the world agree There will be an answer **let it be** For though they may be parted there is still a chance that they will see There will be an answer **let it be**

Let it be let it be let it be let it be There will be an answer **let it be**

Let it be let it be let it be let it be Whisper words of wisdom **let it be**

Let it be let it be let it be let it be Whisper words of wisdom **let it be**

And when the night is cloudy there is still a light that shines on me Shinin until tomorrow **let it be** I wake up to the sound of music Mother Mary comes to me Speaking words of wisdom **let it be**

And **let it be let it be let it be let it be** Whisper words of wisdom **let it be** And **let it be let it be let it be let it be** Whisper words of wisdom **let it be**

Problem 8 [20 points] COMP 1023 Administrative Work

(a) In COMP 1023, your lab total (out of 50) is calculated as follows: 10 points for each lab's attendance and 10 points each for the best 4 out of 5 lab exercises. You are tasked with calculating your lab score using the NumPy library by implementing several functions. The following shows the given main function that calls the functions to be implemented: `attendance()`, `exercise()`, `total()`, and the expected output.

```
import numpy as np

def attendance(scores: np.ndarray) \
    -> tuple[np.ndarray, np.ndarray]:
    # Part (a)(i)

def exercise(scores: np.ndarray) -> np.ndarray:
    # Part (a)(ii)

def total(att: np.ndarray, count: np.ndarray, exe: np.ndarray) \
    -> np.ndarray:
    # Part (a)(iii)

def main() -> None:
    att_scores = np.array([
        # Full
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        # Only half
        [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
        # Excused once + full
        [1, np.nan, 1, 1, 1, 1, 1, 1, 1, 1],
    ])

    att, count = attendance(att_scores)
    print(att, count)

    exercise_scores = np.array([
        [10, 10, 9.5, 9.5, 10],
        [9, 9, 9, 9, 10],
        [10, 10, 10, 10, 10],
    ])

    exe = exercise(exercise_scores)
    print(exe)
    print(total(att, count, exe))

if __name__ == "__main__":
    main()
```

Output:

```
[10.  5.  9.] [10 10  9]
[39.5 37.  40. ]
[ 9.9  8.4 10. ]
```

In the following parts, **you are NOT allowed to use any loops, comprehension or non-NumPy functions**. You may find NumPy's `count_nonzero`, `isnan`, `sort` and `sum` functions useful.

(i) [4 points] With our lab policy, some scores should be excluded from calculation. Implement the function `attendance()`, that calculates students' lab scores and how many should be counted by returning a tuple of **two** 1D arrays in that order. Note that excused lab attendance scores are stored as `np.nan`. You can assume the given array is always a 2D array with 10 columns.

(ii) [3 points] Implement the function `exercise()`, that calculates the sum of the scores of the student's best 4 lab exercises by returning a 1D array. You can assume the given array is always a 2D array with 5 columns.

(iii) [3 points] Using the results of the previous 2 functions, implement the function `total()` to calculate each student's lab total **out of 10** by returning a 1D array. Note that excused labs should be excluded from the lab total calculation, meaning the maximum attainable points are not necessarily 50. You can assume the length of the 3 array parameters are always the same, and the indices always correspond to each other.

(b) Plagiarism is a serious offense. The COMP 1023 teaching team has decided to try out a new way of plagiarism checking, and would like your help in implementing it.

The following is the program that includes all the function headers to be implemented, as well as the main function that utilizes these functions, along with the expected output.

```
import numpy as np

def to_num(c: str) -> int:
    """Converts a SINGLE character into an integer"""
    # Assume this is implemented

def pa_sum(submission: str) -> int:
    # Part (b)(i)

def pairwise(differences: np.ndarray) -> np.ndarray:
    # Part (b)(ii)

def get_students(ids: np.ndarray, pairs: np.ndarray) -> tuple[int, int]:
    # Part (b)(iii)
```

```python
def main():
    ids = [21000000, 21000001, 21000002]
    submissions = ["ABC", "AB", "ZZZ"]
    nums = []

    for s in submissions:
        nums.append(pa_sum(s))
    print(nums) # You do not need to know how the numbers are calculated

    p = pairwise(np.array(nums))
    print(p)
    print(get_students(np.array(ids), p))

if __name__ == "__main__":
    main()
```

Output:

```
[198, 131, 270]
[[  0  67  72]
 [ 67   0 139]
 [ 72 139   0]]
(21000000, 21000001)
```

In the following parts, **you are not allowed to use any loops or comprehensions**. **You are also not allowed to use any non-NumPy functions** in parts (ii) and (iii).

(i) [3 points] The teaching team decided to give each PA submission a score. This score is calculated by first converting each character of a submission into integers, then adding them together. Given the function `to_num()`, which maps any character into an `int`, implement the function `pa_sum()` that sums together a student's converted PA submission **using only ONE statement** (without semi-colons).

(ii) [3 points] The teaching team decides to look at the **absolute** difference in each student's sum as a plagiarism indicator. Implement the function `pairwise()` that calculates the difference between each student's submission **using only ONE statement** (without semi-colons).

(iii) [4 points] Implement the function `get_students()`, where given an array of student IDs, return the pair of students with the minimum pairwise difference in ascending order of student ID. **You can assume that there is only one pair of students with the minimum pairwise difference.**

You may find NumPy's `abs`, `expand_dims`, `fill_diagonal`, `max`, `min`, `newaxis`, `reshape` and `where` useful.

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

Appendix:

Operator Precedence from High (Top) to Low (Down)

Operator	Description	Associativity
`**`	Exponentiation	Right-to-left
`+, -`	Unary plus and minus	Right-to-left
`*, /, //, %`	Multiplication, division, integer division, and modulus	Left-to-right
`+, -`	Binary addition and subtraction	Left-to-right
`<, <=, >, >=`	Relational operators	Left-to-right
`==, !=`	Equality operators	Left-to-right
`is, is not`	Identity operators	Left-to-right
`in, not in`	Membership operators	Left-to-right
`not`	Logical negation	Left-to-right
`and`	Logical conjunction	Left-to-right
`or`	Logical disjunction	Left-to-right
`=, +=, -=, *= /=, //=, %=, **=`	Assignment and augmented assignment operators	Right-to-left*

Python Function Documentation

- Useful functions
  - `dict.get(key, default=None)`
    Return the value for the specified `key` if the key is in the dictionary, otherwise return the specified `default` value (which is `None` if not provided).
  - `dict.items()`
    Return a view object that displays a list of a dictionary's key-value tuple pairs.
  - `dict.keys()`
    Return a view object that displays a list of all the keys in the dictionary.
  - `dict.values()`
    Return a view object that displays a list of all the values in the dictionary.
  - `filter(function, iterable)`
    Construct a new iterable from those elements of `iterable` for which `function` returns true.
  - `len(c)`
    Return the number of items in a container `c`.
  - `list.append(x)`
    Add an item `x` to the end of the list.
  - `list.pop(index)`
    Remove and return the item at the given `index` from the list.

- `map(function, iterable)`
  Apply `function` to every item of `iterable` and return an iterable of the results.
- `sum(iterable)`
  Return the sum of the items in the `iterable`.
- `str.lower()`
  Return a copy of the string with all the characters converted to lowercase.
- `str.split(sep=None)`
  Return a list of the words in the string, using `sep` as the delimiter.

- Pandas functions
  - `pandas.read_csv(filepath, index_col)`
    Read a comma-separated values (CSV) file into a DataFrame. If the `index_col` parameter is 0, it specifies that the first column of the CSV file should be used as the index of the DataFrame.
  - `pandas.DataFrame.max(axis=None)`
    Return the maximum of the values over the requested axis.
  - `pandas.DataFrame.min(axis=None)`
    Return the minimum of the values over the requested axis.

- NumPy functions
  - `numpy.abs(x)`
    Return the absolute value of each element in the array `x`.
  - `numpy.count_nonzero(a)`
    Return the number of non-zero elements in the array `a`.
  - `numpy.expand_dims(a, axis)`
    Expand the shape of an array `a` by inserting a new axis at the specified position.
  - `numpy.fill_diagonal(a, val)`
    Fill the main diagonal of the array `a` with the specified value `val`.
  - `numpy.hstack(tup)`
    Stack arrays in sequence horizontally (column-wise).
  - `numpy.isnan(x)`
    Return a boolean array indicating whether each element of `x` is NaN.
  - `numpy.max(a, axis=None)`
    `numpy.ndarray.max(axis=None)`
    Return the maximum of an array or the maximum along an axis.
  - `numpy.min(a, axis=None)`
    `numpy.ndarray.min(axis=None)`
    Return the minimum of an array or the minimum along an axis.
  - `numpy.nan`
    A floating-point "Not a Number" (NaN) value.

- `numpy.ndarray.T`
  Return the transposed version of the array, which is a view of the array with the axes reversed.
- `numpy.newaxis`
  Used to increase the dimensions of the existing array by one more dimension.
- `numpy.reshape(a, newshape)`
  `numpy.ndarray.reshape(newshape)`
  Give a new shape to an array without changing its data.
- `numpy.sort(a, axis=-1)`
  Return a sorted copy of an array `a`. The default is -1, which sorts along the last axis.
  `numpy.ndarray.sort(axis=-1)`
  Sort an array in-place.
- `numpy.sum(a, axis=None)`
  `numpy.ndarray.sum(axis=None)`
  Return the sum of array elements over a given axis.
- `numpy.where(condition)`
  Return the indices of elements that are non-zero in the input array.

- Matplotlib functions
  - `matplotlib.pyplot.legend()`
    Place a legend on the axes.
  - `matplotlib.pyplot.savefig(fname)`
    Save the current figure to the file `fname`.
  - `matplotlib.pyplot.plot(y, lbl)`
    Plot the values in `y` using the index array $0, 1, \ldots, N-1$ as the x-coordinates, with `lbl` as the label.
  - `matplotlib.pyplot.subplots(figsize=(width, height))`
    Create a new figure of size `width` $\times$ `height` inches and a set of subplots using Matplotlib. It returns a tuple `(fig, ax)`, where the variable `fig` represents the entire figure, and the variable `ax` represents the Axes object(s) where you can plot your data.
  - `matplotlib.pyplot.xticks(angle)`
    Set the angle of the tick labels on the x-axis to `angle` degrees for better visibility.

/* Rough work */

/* Rough work */

/* Rough work */

/* Rough work */