IE UNIVERSITY SCHOOL OF SCIENCE & TECHNOLOGY	**Program:**	**Academic Year:**	**Semester:**
	Course:		
	Year:	**Campus:**	**Group:**
	Type of exam: ☐ Quiz ☐ Midterm ☐ Final ☐ Other		**Date:**
			Num. Pages:

Academic Integrity at IE: cheating, plagiarism, impersonation, falsification, or any dishonest conduct violates the Student Code of Conduct and exam regulations. Such violations may result in failing the exam or losing the call, and repeated or serious misconduct can lead to further sanctions, including expulsion from the program.

Student Name: _____ **Student ID:** _____

Instructions

Always assume each code is run independently, so variables do not persist between exercises.

Questions 1-2: Open Problems (20 points each)

In the following exercises, you will encounter problems that are more open-ended and may not have a single correct answer. Your task is to explain how you would approach each problem by breaking it down into smaller, manageable steps.

Questions 3-8: Multiple Choice (5 points each)

Circle the correct answer for each question, as each question has **only one** correct answer. A correct answer awards 5 points. There is no penalty for wrong choices.

Questions 9-11: Debugging (10 points each)

The functions below contain one or more errors. You have to find these errors, explain why they happen, and suggest a fix to them.

Describe how you would structure your code, which logical steps are needed, and what kind of Python tools or constructs (e.g., loops, conditionals, functions, lists, etc.) you would use. You are not required to write full code, but doing so is encouraged if it helps clarify your ideas.

1. An artificial vision company works with square grayscale images obtained from different satellites. Since each satellite may generate images with different resolutions, the images are stored in a Python list containing NumPy arrays of different sizes. The number of images is unknown beforehand, so they must be processed one by one.

   Each image is analyzed to determine if it is valid. For each image, only the following pixels are considered relevant

   - the pixels on the diagonal,
   - and the top-right and bottom-left corner pixels.

   These values must be extracted and combined into a 1D NumPy array called `relevant_pixels`.

   An image is considered valid if the difference between the maximum and minimum values in `relevant_pixels` does not exceed a `maximum_difference`. Otherwise, the image is considered defective.

   Create a function called `filter_images()` with the following inputs:

   - `images`: a Python list containing square NumPy arrays of different sizes.
   - `maximum_difference`: the maximum allowed difference between relevant pixel values.

   The function must process all images and determine whether each one is valid or defective. Valid images must be stored in a list of image arrays. For each defective image, a message in the format `Incorrect image. Number: {index}` must be added to a separate list, where `{index}` denotes the image position.

   The function returns:

   - a list containing the arrays of all valid images,
   - a list containing the messages corresponding to all defective images.

   If all images are discarded, print a message indicating that no valid images remain.

```
images = [
    np.array([
        [10, 2, 8],
        [4, 9, 6],
        [7, 5, 11]]),

    np.array([
        [1, 20, 3],
        [15, 40, 8],
        [9, 7, 30]])]

maximum_difference = 10

possible_expected_valid_images = [
    np.array([
        [10, 2, 8],
        [4, 9, 6],
        [7, 5, 11]])
]

possible_expected_discarded_images = [ "Incorrect image. Number: 1"]
```

Answer:

The function receives a list of images and the maximum allowed difference, and it must return two lists. Before the loop, we initialize two empty lists: `valid_images = []` and `defective_messages = []`. Because we need both the image and its position, we loop with `for index, image in enumerate(images)`. As the number of images is unknown, a for loop is the right tool. *(+4 points)*

Inside the loop, we extract the relevant pixels of the current image using NumPy. The diagonal is obtained with `np.diag(image)`. The top-right corner is `image[0, -1]` and the bottom-left corner is `image[-1, 0]`, which work for any size because they use negative indices. We combine everything into a single 1-D array with `relevant_pixels = np.concatenate((np.diag(image), [image[0, -1], image[-1, 0]]))`. *(+6 points)*

Then we compute the difference `relevant_pixels.max() - relevant_pixels.min()` and compare it with `maximum_difference`. If it does not exceed it, the image is valid and we append it to `valid_images`. Otherwise, we append the message `f"Incorrect image. Number: {index}"` to `defective_messages`. *(+6 points)*

After the loop, if `valid_images` is empty, we print the message saying that no valid images remain. Finally, we return both lists. *(+4 points)*

(Optional, code)

```python
import numpy as np

def filter_images(images: list[np.ndarray], maximum_difference: float) -> tuple:
    valid_images = []
    defective_messages = []

    for index, image in enumerate(images):
        # Diagonal + top-right corner + bottom-left corner
        relevant_pixels = np.concatenate((np.diag(image),
                                          [image[0, -1], image[-1, 0]]))

        if relevant_pixels.max() - relevant_pixels.min() <= maximum_difference:
            valid_images.append(image)
        else:
            defective_messages.append(f"Incorrect image. Number: {index}")

    if len(valid_images) == 0:
        print("No valid images remain.")

    return valid_images, defective_messages
```

2. You are an English teacher who wants to automatically evaluate a speaking exam. Each student answered several questions, and their answers are stored in a dictionary where each key is the student's name and each value is a list of string answers. (**See the example below**)

   To grade the exam, a list of keyword expressions will be provided. For each student, the program must **check** every answer and **count** how many times each keyword appears (Keyword matching should be case-insensitive). The results must be stored in a new **nested dictionary**, where the key is the student's name and the value is another dictionary containing the number of times each keyword was used and the total score obtained by that student.

   A student passes the exam only if their total score is greater than or equal to a given passing threshold. Students who do not reach the threshold must be considered failed and excluded from the final dictionary.

   Finally, the program must return the final dictionary.

   Create a function called `evaluate_exam()` to perform this task. The function must take the following inputs:

   - `students`: a dictionary where each key is the name of a student and each value is a list of string answers.
   - `keywords`: a list of keyword expressions used for grading.
   - `passing_score`: the minimum score required to pass the exam.

```python
students = {
    "Student1": ["Personally English is very useful.",
        "Fortunately I can practice every day."],

    "Student2": [ "I like sports.",
        "My favourite hobby is reading."],

    "Student3": [ "Obviously technology is important.",
        "Furthermore it helps communication."]
}

keywords = ["personally","fortunately","obviously","furthermore","
    communication"]

passing_score = 1

# Output:
{
    'Student1': {
        'personally': 1,
        'fortunately': 1,
        'total_score': 2},

    'Student3': {
        'obviously': 1,
        'furthermore': 1,
        'communication': 1,
        'total_score': 3}
}
```

Answer:

The function loops through the dictionary of students and, for each student, through each of their answers. As the result is a new dictionary, before the loop we initialize `result = {}`. The outer loop is `for name, answers in students.items()`, and the inner one is `for answer in answers`. *(+4 points)*

For each student we create an empty dictionary `counts = {}` and a variable `total = 0`. To make the matching case-insensitive, we first lowercase the answer, `text = answer.lower()`. Then, with a third loop over `keywords`, we count the appearances of each keyword using the string method `text.count(keyword.lower())`. *(+6 points)*

If the count is greater than zero, we accumulate it in the dictionary of that student, `counts[keyword] = counts.get(keyword, 0) + n`, and add it to the `total`. We use `.get(keyword, 0)` because a keyword can appear in several answers of the same student, and the key may not exist yet. Only the keywords that were actually used appear in the output, as in the example. *(+5 points)*

When we finish with the answers of a student, we check if `total >= passing_score`. If so, we store `counts["total_score"] = total` and add `counts` to the result under the student name. Students who do not reach the threshold are simply not added. Finally, after the loops, we return `result`. *(+5 points)*

(Optional, code)

```python
def evaluate_exam(students: dict, keywords: list, passing_score: int) -> dict:
    result = {}

    for name, answers in students.items():
        counts = {}
        total = 0

        for answer in answers:
            text = answer.lower()
            for keyword in keywords:
                n = text.count(keyword.lower())
                if n > 0:
                    counts[keyword] = counts.get(keyword, 0) + n
                    total += n

        # Only students that pass are included
        if total >= passing_score:
            counts["total_score"] = total
            result[name] = counts

    return result
```

3. After running the following code, what will be the value of "A"?

```
import numpy as np

A = np.array([[0, 1, -1],
              [0, 1, 1],
              [1, -1, -1]])

B = A[1: , 1:]

mask = B < 0

A[mask] = 0

print(A)
```

(a) $\begin{pmatrix} 0 & 1 & -1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$

(b) $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$

(c) $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

(d) The code will raise an error because the mask does not have the same shape as array A.

4. After running the following code, what will be the value of "warning"?

```
warning = "Reminder: You are scheduled to "
task = ("play padel", "May 15", "9:00 PM")

def create_warning(warning: str)->str:
  warning = warning + task[0] + " on " + task[1] + " at " + task[2]

create_warning(warning)

print(warning)
```

(a) `warning = ''Reminder: You are scheduled to play padel on May 15 at 9:00 PM''`

(b) `warning = ''Reminder: You are scheduled to ''`

(c) The code raises a `NameError` because `task` is not included as a parameter of the function.

(d) The code raises a `TypeError` because strings are immutable and cannot be concatenated.

5. After running this code, what will be the final value of "tour"?

```
dest_1 = ["Japan", "Spain", "Italy", "Malaysia", "Sri Lanka"]

dest_2 = ["France", "Sri Lanka", "Japan", "Australia"]

dest_3 = ["Australia", "Sri Lanka", "Japan"]

dest_list = sorted(list(set(dest_1) & set(dest_2) | set(dest_3)))

tour = " , ".join([f"{i}.{dest}" for i, dest in enumerate(dest_list,1)])

print(tour)
```

   (a) `"1.Japan , 2.Sri Lanka"`

   (b) `"1.Australia , 2.Japan , 3.Sri Lanka"`

   (c) `[(1, Australia), (2, Japan), (3, Sri Lanka)]`

   (d) The code will raise an error.

6. Why will the following code fail?

```
import numpy as np

rng = np.random.default_rng(42)

arr_1 = np.arange(1, 10).reshape((3, -1))
arr_2 = rng.choice([1, 10, 20, 30, 40, 50], size=(3, 3))

if arr_1 == arr_2:
    print("Both matrices have some equal values")
else:
    print("They are completely different")
```

   (a) Because `arr_1 == arr_2` returns a Boolean array, and an `if` statement requires a single Boolean value.

   (b) Because `(3, -1)` and `(3, 3)` are not the same shape, so the arrays cannot be compared.

   (c) Because `np.arange(1, 10)` creates 10 elements, which cannot be reshaped into a `(3, 3)` array.

   (d) Because `rng.choice()` cannot generate a two-dimensional array.

7. After running this code, what will be the final value of "result"?

```
init_data = [1, -2, 4, -3, 5, -1]

result = [x**2 if x - 2 > 0 else (x + 2)**2 for x in init_data if x > 0]

print(result)
```

(a) `[1, 16, 25]`

(b) `[9, 0, 16, 1, 25, 1]`

(c) `[9, 16, 25]`

(d) The code will raise an error.

8. What will be the value of array `B` after applying the function `modify_matrix()`?

```
import numpy as np

def modify_matrix(arr: np.ndarray)->np.ndarray:

    row_vector = np.array([0, 1, -1])

    column_vector = row_vector.reshape((-1, 1))

    C = arr + row_vector

    D = C*column_vector

    return D

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

B = modify_matrix(A)
```

(a) $\begin{pmatrix} 0 & 0 & 0 \\ 4 & 6 & 5 \\ -7 & -9 & -8 \end{pmatrix}$

(b) $\begin{pmatrix} 0 & 2 & -3 \\ 0 & 6 & -7 \\ 0 & 7 & -8 \end{pmatrix}$

(c) $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 1 \\ -7 & -8 & -9 & -1 \end{pmatrix}$

(d) The code will raise an error because the dimensions are not compatible for broadcasting.

Answers: 3-d, 4-b, 5-b, 6-a, 7-c, 8-a

9. The following code is intended to filter and rank a list of hotels for a specific city in order to make it easier for users to decide which one to choose. The hotel list contains sublists, where the first element is the hotel name, the second element is the user rating, and the third element is the number of reviews.

   First, the program must filter the list by removing all hotels whose user rating is lower than 8.5. Then, the program must sort the filtered list in descending order of user rating and, in case of ties, in descending order of number of reviews.

   However, the code does not produce the desired output. Some hotels with a user rating lower than 8.5 remain in the filtered list, and the list is not sorted correctly.

   Identify the two errors and correct them.

```
hotels = [
    ["Grand Palace Hotel", 7.3, 120], ["Inn City Center", 7.4, 85],
    ["Royal Gardens", 9.1, 450], ["Budget Stay", 9.1, 180],
    ["Sunset Resort", 8.2, 320], ["Green Park Hotel", 7.4, 210],
    ["Riverside Lodge", 8.6, 275] ]

# Iterate through the hotels list
for hotel in hotels:
    # Check if the user rating is lower than 8.5
    if hotel[1] < 8.5:
        hotels.remove(hotel)

# Sort the hotels list by user rating and then by number of reviews
hotels.sort(key=lambda hotel: hotel[0], reverse=True)

#Expected output
#[['Royal Gardens', 9.1, 450], ['Budget Stay', 9.1, 180], ['Riverside
   Lodge', 8.6, 275]]

#Actual output
#[['Royal Gardens', 9.1, 450], ['Riverside Lodge', 8.6, 275], ['Inn City
    Center', 7.4, 85], ['Green Park Hotel', 7.4, 210], ['Budget Stay',
   9.1, 180]]
```

Answer:

The code has two errors. The first one is that the for loop removes elements from the list `hotels` while iterating over it. When an element is removed, the following elements shift one position to the left, and the loop skips the next hotel. That is why some hotels with a rating lower than 8.5 (`Inn City Center` and `Green Park Hotel` in the example) remain in the list. We should not modify a list while looping through it. Instead, we can build a new list with a list comprehension that keeps only the hotels we want, `[hotel for hotel in hotels if hotel[1] >= 8.5]`, or loop over a copy, `for hotel in hotels.copy()`. *(+5 points)*

The second error is the sorting key. The lambda uses `hotel[0]`, which is the hotel name, so the list is sorted alphabetically in reverse order. We need to sort by rating first and, in case of ties, by number of reviews. The key must return a tuple with both values, `(hotel[1], hotel[2])`. With `reverse=True`, both criteria are descending. *(+5 points)*

(Optional, fixed code)

```
hotels = [
    ["Grand Palace Hotel", 7.3, 120], ["Inn City Center", 7.4, 85],
    ["Royal Gardens", 9.1, 450], ["Budget Stay", 9.1, 180],
    ["Sunset Resort", 8.2, 320], ["Green Park Hotel", 7.4, 210],
    ["Riverside Lodge", 8.6, 275] ]

# Keep only hotels with a rating of at least 8.5
hotels = [hotel for hotel in hotels if hotel[1] >= 8.5]

# Sort by rating and then by number of reviews, both descending
hotels.sort(key=lambda hotel: (hotel[1], hotel[2]), reverse=True)
```

10. This code is intended to process a matrix of coefficients obtained from a numerical simulation. Only the coefficients that satisfy the condition

$$x^2 + x - 20 > 0$$

should be copied from A into a new matrix B initially filled with ones; all other positions should remain as ones. (see the expected output). The function raises the following error on line 16: "`IndexError: too many indices for array: array is 1-dimensional, but 2 were indexed`". Identify the two errors and correct them.

```
1  import numpy as np
2
3  A = np.array([
4      [2, 4, 6, 10],
5      [3, 5, 7, 12],
6      [1, 2, 15, 18]
7  ])
8
9  #Create an array of ones
10 B = np.ones(A.shape[0])
11
12 #Create a mask containing the elements that satisfy the condition
13 mask = A**2 + A - 20 > 0
14
15 #Copy the valid elements from A into B
16 B[mask] = A
17
18 print(B)
19
20 #Expected output
21 #[[ 1.  1.  6. 10.]
22 #[ 1.  5.  7. 12.]
23 #[ 1.  1. 15. 18.]]
```

Answer:

The first error is on line 10, where the array of ones is created with `np.ones(A.shape[0])`. This only uses the number of rows, so `B` is a 1-D array of length 3, while the expected output is a 2-D matrix with the same shape as `A`. The correct code is `B = np.ones(A.shape)`. Because `B` is 1-D, indexing it with the 2-D mask is what raises the `IndexError` of line 16. *(+5 points)*

The second error is on line 16, `B[mask] = A`. The left side selects only the positions where the mask is True, but the right side is the full matrix `A`, so the shapes do not match. We must copy only the elements of `A` that satisfy the condition: `B[mask] = A[mask]`. *(+5 points)*

(Optional, fixed code)

```
1  import numpy as np
2
3  A = np.array([
4      [2, 4, 6, 10],
5      [3, 5, 7, 12],
6      [1, 2, 15, 18]
7  ])
```

```
8
9  # Create an array of ones with the same shape as A
10 B = np.ones(A.shape)
11
12 # Create a mask containing the elements that satisfy the condition
13 mask = A**2 + A - 20 > 0
14
15 # Copy the valid elements from A into B
16 B[mask] = A[mask]
17
18 print(B)
```

11. This code is intended to rearrange a matrix with an even number of columns. The matrix is split into two equal column-wise halves, each half is flattened into a 1-D array, and the resulting arrays are stacked vertically to form a new 2-D array.

    The function works correctly for matrix `B`, but it raises an error for matrix `C` on line 17 (see below). Identify the error and correct it.

```
1  import numpy as np
2
3  def rearrange_matrix(A: np.array) -> np.array:
4      # Compute the number of columns
5      columns = A.shape[1]
6
7      # Check that A has an even number of columns
8      if columns % 2 == 0:
9
10         # Split the matrix into its left and right halves
11         left_A = A[:, :2]
12         right_A = A[:, 2:]
13
14         # Flatten both submatrices
15         flat_l = left_A.flatten()
16         flat_r = right_A.flatten()
17
18         # Stack the flattened arrays into a 2D array
19         return np.vstack((flat_l, flat_r))
20
21     print("Only valid for 2D matrices with an even number of columns")
22
23 B = np.array([
24     [1, 2, 3, 4],
25     [5, 6, 7, 8]
26 ])
27
28 print(rearrange_matrix(B))
29
30 #Output
31 #[[1 2 5 6]
32 # [3 4 7 8]]
33
34 C = np.array([
35     [1, 2, 3, 4, 5, 6],
36     [7, 8, 9, 10, 11, 12],
37     [13, 14, 15, 16, 17, 18]
38 ])
39
40 print(rearrange_matrix(C))
41
42 # ValueError: arrays have incompatible dimensions for concatenation.
43 # Along dimension 1, the array at index 0 has size 6 and the array at
44 # index 1 has size 12.
```

Answer:

The error comes from the lines where the matrix is split. The columns are split at the hard-coded position 2: `left_A = A[:, :2]` and `right_A = A[:, 2:]`. This works for `B`, which has 4 columns, but not for `C`, which has 6: the left half takes 2 columns and the

right half takes 4, so the flattened arrays have different lengths (6 and 12), and `np.vstack` cannot stack them. *(+5 points)*

The split position must be the middle of the matrix, which depends on the number of columns. We compute it with the integer division `half = columns // 2` and use it in both slices: `A[:, :half]` and `A[:, half:]`. *(+5 points)*

(Optional, fixed code)

```python
import numpy as np

def rearrange_matrix(A: np.array) -> np.array:
    # Compute the number of columns
    columns = A.shape[1]

    # Check that A has an even number of columns
    if columns % 2 == 0:
        # Position where the matrix is split in two halves
        half = columns // 2

        # Split the matrix into its left and right halves
        left_A = A[:, :half]
        right_A = A[:, half:]

        # Flatten both submatrices
        flat_l = left_A.flatten()
        flat_r = right_A.flatten()

        # Stack the flattened arrays into a 2D array
        return np.vstack((flat_l, flat_r))

    print("Only valid for 2D matrices with an even number of columns")
```