IE UNIVERSITY
SCHOOL OF SCIENCE & TECHNOLOGY

	Program:	**Academic Year:**
Course:		
	Year:	**Campus:**
Type of exam: ☐ Quiz ☐ Midterm ☐ Final ☐ Other		

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)

These are open-ended tasks. Explain your approach, outline algorithmic steps, and optionally provide compact working code snippets.

Questions 3-5: Debugging (10 points each)

These exercises show code cells with one or more errors. You have to find these errors, explain why they happen, and suggest a fix to them.

Questions 6-11: Multiple Choice (5 points each)

On the table next page, mark the correct answer for each question. Use an X to mark your choice. If you want to change your answer, fill the entire box and mark a new one. If multiple boxes are marked for the same question, no points will be awarded. See the examples below. Each question has **only one** correct answer. A correct answer awards 5 points. There is no penalty for wrong choices.

Example	A	B	C	D
Chosen A	X	☐	☐	☐
Chosen C, cancelled A	■	☐	X	☐
Chosen B, cancelled A and C	■	X	■	☐
Invalid (multiple choices)	X	X	☐	☐
Invalid (no choices)	■	☐	☐	☐

Oral Justification. Students may be asked to provide oral justification for any of the exercises after completing the written portion.

Mark your answers for Questions 6–11 below

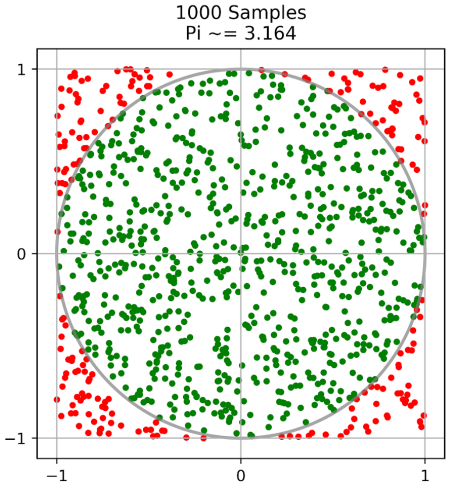
Question	A	B	C	D
Q6	☐	☐	☐	☐
Q7	☐	☐	☐	☐
Q8	☐	☐	☐	☐
Q9	☐	☐	☐	☐
Q10	☐	☐	☐	☐
Q11	☐	☐	☐	☐

1. You want to approximate π using two methods: the Leibniz formula and the Monte Carlo method.

   The **Leibniz method** represents π through an alternating infinite series made of odd denominators. If you stop after a finite number of terms, you obtain an approximation. As more terms are included, the approximation oscillates around π and gradually improves:

$$\pi \approx 4 \times \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$$

   The **Monte Carlo method** uses geometric probability. Imagine a square of side length 2 centered at the origin, with an inscribed circle of radius 1. If points are uniformly scattered across the square, the fraction that falls inside the circle is approximately equal to the ratio between the area of the circle and the area of the square.



   Using this area-ratio idea, π can be approximated as:

$$\pi \approx 4 \times \frac{\text{points inside circle}}{\text{total points}}$$

   Tip: a point i is inside the circle if $x_i^2 + y_i^2 \leq 1$, where (x_i, y_i) are its coordinates.

   Describe how you would:

   - translate both ideas into Python functions, *(8 points each)*
   - test how close the approximations get to the true value of π (assume you have it in a variable called `pi_ref`). *(4 points)*

Answer:

We want to implement two functions, one deterministic (Leibniz) and one stochastic (Monte Carlo), and then compare their behavior as computation effort increases. One possible function signatures are:

```
def leibniz_pi(n: int) -> float
def monte_carlo_pi(num_points: int, seed: int = 314) -> float
```

For the Leibniz method, we can use a simple loop over k from 0 to n, compute each term of the series, and accumulate the sum on a variable `result` that we would initially set to 0 before the loop. Finally, we multiply by 4 to get the approximation. *(8 points)*

For the Monte Carlo method, we use NumPy to generate `num_points` random coordinates (x, y) uniformly distributed in the square [-1, 1] x [-1, 1]. This can be done with `x = np.random.uniform(-1, 1, num_points)` and same for y. We then count how many points fall inside the unit circle with $x^2 + y^2 \leq 1$, easy with broadcasting. This gives a boolean array that we can sum to get the count of points inside the circle. Finally, we compute the approximation as $4 \times$ inside/num_points. We can also set a random seed for reproducibility. *(8 points)*

To test the approximations, we can compute the absolute error with respect to `pi_ref` for different values of n and `num_points`. In Python, this is simply `abs(approximation - pi_ref)`. We can run a loop over increasing values of n and `num_points` to see how the error decreases for both methods. *(4 points)*

Full implementation example:

```python
import numpy as np

def leibniz_pi(n: int) -> float:
    result = 0.0
    for k in range(n + 1):
        result += ((-1) ** k) / (2 * k + 1)
    return 4 * result

def monte_carlo_pi(num_points: int, seed: int = 314) -> float:
    rng = np.random.default_rng(seed)
    x = rng.uniform(-1, 1, num_points)
    y = rng.uniform(-1, 1, num_points)
    inside = np.sum(x**2 + y**2 <= 1)
    return 4 * inside / num_points

# Example of testing the approximations
pi_ref = np.pi
for n in [10, 100, 1000, 10000]:
    pi_approx = leibniz_pi(n)
    error = abs(pi_approx - pi_ref)
    print(f"Leibniz n={n}: approximation={pi_approx}, error={error}")

for num_points in [100, 1000, 10000]:
    pi_approx = monte_carlo_pi(num_points)
    error = abs(pi_approx - pi_ref)
    print(f"Monte Carlo num_points={num_points}: approximation={pi_approx
   }, error={error}")
```

2. A language school assesses four oral-test skills with different weights in the final score: fluency (35%), grammar (20%), vocabulary (15%), and pronunciation (30%). Each student receives a score from 0 to 10 for each skill. For instance:

Student	Fluency (35%)	Grammar (20%)	Vocabulary (15%)	Pronunciation (30%)
Alice	8.5	7.0	6.5	9.0
Bob	5.0	4.5	5.5	6.0
Charlie	9.0	8.5	7.0	9.5

The school wants to organize the data in Python, in a way that allows easy access to each student's scores. They don't want to use external libraries (such as NumPy or Pandas). The school also want to compute each student's weighted total score, find out who failed (total score below 5), and who deserves a certificate of excellence (total score above 9).

Your answer must explain:

- how you would organize the input data, *(6 points)*
- how you would compute each student's weighted total score and store it, *(8 points)*
- how you would you find which students failed (total score below 5) and which ones deserve a certificate of excellence (total score above 9), *(6 points)*

Answer:

The most convenient input structure is a dictionary whose keys are student names and whose values are dictionaries with the four skill scores. For instance: `scores = {"Alice": {"fluency": 8.5, "grammar": 7.0, "vocabulary": 6.5, "pronunciation": 9.0}, ...}`. The output can be another dictionary storing each student's weighted total, or a new key in the existing dictionary for each student. *(6 points)*

To compute the total, we can loop through the dictionary keys (student names) using `for name in scores.keys()`. For each student, calculate the weighted sum of their scores. A simple approach would be `total = 0.35 * scores[name]["fluency"] + 0.2 * scores[name]["grammar"] + ...` assuming all students have the full set of categories. We could store the result in the same dictionary, as they are mutable: `scores[name]["total"] = total`. *(8 points)*

In the same loop, we can check if the total is below 5 or above 9 and append the student's name to separate lists for failed and excellent students. Those lists can be initialized as empty before the loop and then updated with `failed.append(name)` or `excellent.append(name)` accordingly. *(6 points)*

One possible implementation is:

```
def compute_results(scores: dict) -> tuple:
    totals = {}
    failed = []
    excellent = []

    for name in scores.keys():
        fluency = scores[name]["fluency"]
        grammar = scores[name]["grammar"]
        vocabulary = scores[name]["vocabulary"]
        pronunciation = scores[name]["pronunciation"]

        total = 0.35 * fluency + 0.2 * grammar + 0.15 * vocabulary + 0.3
   * pronunciation
        totals[name] = total

        if total < 5:
            failed.append(name)
        elif total > 9:
            excellent.append(name)
    return totals, failed, excellent
```

Another, more professional approach, would allow the user to easily change the weights without modifying the function logic.

```
weights = {
        "fluency": 0.35,
        "grammar": 0.2,
        "vocabulary": 0.15,
        "pronunciation": 0.3
    }

def compute_results(scores: dict, weights: dict) -> tuple:
    totals = {}
    failed = []
    excellent = []

    for name in scores.keys():
        total = 0
        for skill, weight in weights.items():
            # Safe access in case some students
            # are missing some skill scores
            if skill in scores[name].keys():
                total += weight * scores[name][skill]
            # If the skill is missing, it will count as zero,
            # so we can just skip it
        totals[name] = total

        if total < 5:
            failed.append(name)
        elif total > 9:
            excellent.append(name)
    return totals, failed, excellent
```

3. This code is intended to create a 2-D array A from an array B with 2 columns and an odd number of rows, placing the two columns of B into A in the shape of a cross (see the expected output). The function raises an error on line 20: "`ValueError: too many indices for array: array is 1-dimensional, but 2 were indexed`". Identify the three errors and correct them.

```
1  import numpy as np
2
3  def cross_matrix(B: np.ndarray) -> np.ndarray:
4    # Get number of rows
5    n = B.shape[1]
6    # Create an array filled with zeros
7    A = np.zeros(n)
8    # Extract both columns
9    column1 = B[0, :]
10   column2 = B[:, 1]
11   # Add the columns as a cross to the new array
12   A[:, n // 2] = column1
13   A[n // 2, :] = column2
14   return A
15
16 B = np.array([[1, 4],
17                [2, 5],
18                [3, 6]])
19
20 print(cross_matrix(B))
21 # Expected output
22 # [[0. 1. 0.]
23 #  [4. 5. 6.]
24 #  [0. 3. 0.]]
```

Answer:

The first error is on line 5, where we get the number of rows with `B.shape[1]`. This actually gives us the number of columns. The correct way to get the number of rows is `n = B.shape[0]`. *(3 points)*

The second error is on line 7, where we create the array `A` with `np.zeros(n)`. This creates a 1-D array of length n, but we need a 2-D array of shape (n, n) to place the columns in a cross shape. The correct code is `A = np.zeros((n, n))`. *(3 points)*

The third error is the indexing on line 9. The expression `B[0, :]` tries to access the first row of B, but we want the first column. The correct way to get the first column is `B[:, 0]`. Similarly, for the second column, it should be `B[:, 1]`. *(4 points)*

4. The following code is intended to replace all values greater than 6 in `arr` with `-1`. However, the printed matrix does not change. Explain why and fix the code.

```
import numpy as np

arr = np.array([
    [2, 8, 1],
    [9, 3, 7],
    [4, 6, 10]
])

high = arr[arr > 6]
arr[high] = -1

print(arr)
# Expected:
# [[ 2 -1  1]
#  [-1  3 -1]
#  [ 4  6 -1]]
# Current output:
# IndexError: index 8 is out of bounds for axis 0 with size 3
```

Answer:

The issue is that `high` is not a boolean mask, but an array of the values that are greater than 6. When we do `arr[high] = -1`, it tries to access the indices of `arr` corresponding to those values, which leads to an index error because those values are not valid indices. *(5 points)*

To fix, simply use the boolean mask directly to assign `-1` to the correct positions in `arr`. The corrected line should be: `arr[arr > 6] = -1`. This way, we are directly modifying the elements of `arr` that satisfy the condition. *(5 points)*

Here is the corrected code:

```
import numpy as np

arr = np.array([
    [2, 8, 1],
    [9, 3, 7],
    [4, 6, 10]
])

arr[arr > 6] = -1

print(arr)
```

5. The following code is intended to remove consecutive duplicated values from a list (for example, `[3,3,1,1,2]` should become `[3,1,2]`). However, it raises an error. Identify the one problem and suggest a fix.

```
values = [3, 3, 1, 1, 2, 2, 2, 5]

idx = 0
while idx < len(values):
    if values[idx] == values[idx + 1]:
        values.pop(idx + 1)
    else:
        idx += 1

print(values)
# Expected output: [3, 1, 2, 5]
# Current output (in line 5):
# IndexError: list index out of range
```

Answer:

The problem is an index boundary error: inside the `while` loop the code compares `values[idx]` with `values[idx + 1]`, but when `idx` reaches the last valid position, `idx + 1` is out of range and raises an `IndexError`. *(5 points)*

To fix this, the loop condition must stop at the second-to-last element, and we should only increase `idx` when no deletion happens. If a deletion happens, we keep the same index to check the new neighbor pair again. *(5 points)*

Here is the corrected code:

```
values = [3, 3, 1, 1, 2, 2, 2, 5]

idx = 0
while idx < len(values) - 1:
    if values[idx] == values[idx + 1]:
        values.pop(idx + 1)
    else:
        idx += 1

print(values)  # [3, 1, 2, 5]
```

6. After running the following code, what kind of numbers will `L` contain?

```
import numpy as np

A = np.arange(1, 13).reshape((3, 4))
L, R = np.split(A, 2, axis=1)
```

A. Only even numbers.

B. Only odd numbers.

C. The code raises an error because the split is not possible with the given shape.

D. Both even and odd numbers.

7. Being x_i the elements of the array `x`, what is the value of z_i after running this code?

```
import numpy as np

x = np.array([3, 11, 6, 7, 4, 8])
xmin = np.min(x)
xmax = np.max(x)
xcut = x - xmin
z = xcut / xmax - xmin
```

A. $z_i = (x_i - 36)/11$

B. $z_i = (x_i - 3)/11$

C. The code will fail because it is mixing arrays and scalars in the operations.

D. $z_i = (x_i - 3)/8$

8. What is the value of `B`?

```
import numpy as np

A = np.array([
    [1, 4],
    [7, 2]
])

B = np.where(A > 3, A, -A)
```

A. $\begin{pmatrix} -1 & 4 \\ 7 & -2 \end{pmatrix}$

B. $\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$

C. The code raises an error because `np.where` cannot be used with matrices.

D. $\begin{pmatrix} 1 & -4 \\ -7 & 2 \end{pmatrix}$

9. Consider:

```
a = {"python", "numpy", "pandas"}
b = {"numpy", "sql"}
c = {"python", "sql"}
result = (a & b) | c
```

What is `result`?

A. `{"python", "sql"}`

B. `{"numpy"}`

C. `{"python", "numpy", "pandas", "sql"}`

D. `{"python", "numpy", "sql"}`

10. After running this code, what is the final value of `y`?

```
def add_city(city, bucket=[]):
    bucket.append(city)
    return bucket

x = add_city("Rome")
y = add_city("Paris")
```

A. The code raises an error because default values cannot be lists.

B. `["Rome", "Paris"]`

C. `["Paris"]`

D. `["Rome"]`

11. Which statement about Python functions is FALSE?

A. If a function has no explicit `return`, it returns `None`.

B. Using `print()` inside a function makes that printed value the function's return value.

C. A function can read a global variable if there is no local variable with the same name.

D. A variable created inside a function is local to that function unless declared global.

Correct answers for Questions 6–11:

Question	Key
??	??
??	??
??	??
??	??
??	??
??	??