# Sheet 01 — Problems (Labs 01–03)

Department of Mechatronics & Robotics Engineering
Egyptian Russian University

## Problem 1 — Robust Greeting (Simple)

Write a program that:

1. Reads the user's full name (may include multiple spaces) and their birth year.
2. Validates birth year is a 4-digit positive integer; re-prompt until valid.
3. Prints: `Hello <Name>, you are <age> years old.`

## Problem 2 — Weighted Average (Simple → Medium)

Read five exam scores (0–100) and five corresponding weights (positive, sum not necessarily 1). Compute the weighted average normalized to 100 and print it to two decimals. Validate inputs (scores in range, weights positive).

## Problem 3 — Grade Classification with Ties (Medium)

Ask for $n$ student averages (float). For each average print letter grade using:

- A: $\geq 90$
- B: [80,90)
- C: [70,80)
- D: [60,70)
- F: ¡60

Then print how many students received each letter. Use functions for classification and counting.

## Problem 4 — Sliding Window Max (Medium)

Read an integer list from user (space separated). Then read window size $k$ (integer, $1 \leq k \leq len(list)$). Compute and print the maximum for each sliding window of length $k$. (Implement with simple loops, not collections.deque.)

**Example:** list = [2,1,3,4,6,3,8,9,10,12,56], k=4 ⇒ outputs maxima per window.

### Problem 5 — Token Frequency & Top-3 Words (Medium → Above)

Read a paragraph. Normalize to lowercase, remove punctuation ‘. , ; : ! ? ( ) ” ’‘, split into words, count word frequencies, and print the top-3 most frequent words and counts. Use functions for tokenization and counting.

### Problem 6 — Prime Factorization (Above Medium)

Write a function `prime_factors(n)` that returns a list of (prime, exponent) pairs for integer $n > 1$. Input an integer $n$ and print its prime factorization in the format: `n = p1â1 * p2â2 * ...`. Use only basic loops.

### Problem 7 — Matrix Multiplication (Above Medium)

Read two matrices A (size m x p) and B (size p x n) from user (first line: m p, then m lines; then p n, then p lines). Multiply A and B and print result matrix. Use nested lists and nested loops.

### Problem 8 — Mini Project: CLI Gradebook with Functions (Above Medium)

Build a small menu-driven gradebook that repeatedly allows the user to:

(a) Add student: enter name and three exam grades (validate).

(b) Show student report: for each student print name, grades, average, letter grade.

(c) Class statistics: class average, highest average with name, lowest average with name.

(d) Exit.

Implement at least these functions: `add_student()`, `compute_average(marks)`, `letter_grade(avg)`, `class_statistics(students)`. Use lists of dictionaries or parallel lists.

---

## Model Answers (Detailed, line-commented)

**Note:** Comments explain the syntax or algorithm step. These are model solutions — students may implement differently.

### Solution 1 — Robust Greeting

```
# Solution 1
# input() returns a string
name = input("Enter your full name: ")
# read as string for validation
year_str = input("Enter your birth year (YYYY): ")
while not (year_str.isdigit() and len(year_str) == 4 and int(
    year_str) > 0):
```

```
# isdigit() checks all chars are digits; len==4, format YYYY
  year_str = input("Invalid. Enter birth year (YYYY): ")
birth_year = int(year_str)    # convert validated string to int
# arithmetic to compute age (use current year or parameterize)
age = 2025 - birth_year
# f-string formats the output
print(f"Hello {name}, you are {age} years old.")
```

## Solution 2 — Weighted Average

```
# Solution 2
# Prepare lists with 5 placeholders
scores = [0] * 5
weights = [0] * 5

# Input exam scores (0-100)
i = 0
while i < 5:
    s = float(input(f"Enter score #{i+1} (0-100): "))
    while not (0 <= s <= 100):  # validate score
        s = float(input("Invalid score. Enter 0-100: "))
    scores[i] = s  # store by index
    i += 1

# Input weights (positive)
i = 0
while i < 5:
    w = float(input(f"Enter weight #{i+1} (positive): "))
    while w <= 0:  # validate weight
     w = float(input("Invalid weight. Enter positive value: "))
    weights[i] = w  # store by index
    i += 1

# Compute weighted sum manually
weighted_sum = 0
i = 0
while i < 5:
 weighted_sum += scores[i] * weights[i]
 i += 1

# Compute sum of weights manually
norm = 0
i = 0
while i < 5:
 norm += weights[i]
 i += 1

# Compute weighted average normalized to 100
weighted_avg = (weighted_sum / norm)  # Already normalized to
   weight sum
```

```
# Print result with 2 decimals
print(f"Weighted average = {weighted_avg:.2f}")
```

## Solution 3 — Grade Classification with Ties

```
# Solution 3
def classify(avg):    # function to return letter grade
if avg >= 90:
return 'A'
elif avg >= 80:
return 'B'
elif avg >= 70:
return 'C'
elif avg >= 60:
return 'D'
else:
return 'F'

n = int(input("Number of students: "))    # number of inputs
# dictionary to count grades
counts = {'A':0,'B':0,'C':0,'D':0,'F':0}
for i in range(n):
# read average
avg = float(input(f"Enter average for student {i+1}: "))
g = classify(avg)          # call classification function
print(f"Grade: {g}")       # output letter
counts[g] += 1             # increment count

print("Grade counts:")
for k in ['A','B','C','D','F']:
print(k, counts[k])       # print counts per letter
```

## Solution 4 — Sliding Window Max

```
# Solution 4
data = list(map(int, input("Enter integers (space separated): "
    ).split()))
k = int(input("Enter window size k: "))
n = len(data)
if not (1 <= k <= n):
# simple error exit
print("Invalid window size"); raise SystemExit

# compute max for each window starting at i..i+k-1
for i in range(0, n - k + 1):
window_max = data[i]      # start with first element in window
for j in range(i+1, i+k):     # check remaining
if data[j] > window_max:
window_max = data[j]
```

```
print(f"Window {i+1} max: {window_max}")
```

## Solution 5 — Token Frequency & Top-3 Words

```
# Solution 5
import string   # string.punctuation contains punctuation chars
def tokenize(text):
text = text.lower()              # normalize to lowercase
for ch in string.punctuation:    # remove punctuation
text = text.replace(ch, ' ')
tokens = text.split()            # split on whitespace
return tokens

paragraph = input("Enter paragraph: ")
tokens = tokenize(paragraph)
freq = {}                        # dictionary for counts
for t in tokens:
freq[t] = freq.get(t, 0) + 1  # increment count (get default 0)

# get top 3 frequent words
# sort by count desc
items = sorted(freq.items(), key=lambda x: x[1], reverse=True)
top3 = items[:3]
print("Top-3 words:")
for word, count in top3:
print(word, count)
```

## Solution 6 — Prime Factorization

```
# Solution 6
def prime_factors(n):
factors = []
d = 2
while d * d <= n:       # test divisors up to sqrt(n)
exp = 0
while n % d == 0:       # count how many times d divides n
n //= d
exp += 1
if exp > 0:
factors.append((d, exp))  # record prime and exponent
# move: 2 -> 3, then skip even numbers
d += 1 if d == 2 else 2
if n > 1:
factors.append((n, 1))    # remaining prime factor
return factors

num = int(input("Enter integer (>1): "))
pf = prime_factors(num)
# format output
```

```
    parts = [f"{p}^{e}" for (p, e) in pf]
    print(str(num) + " = " + " * ".join(parts))
```

## Solution 7 — Matrix Multiplication

```
# Solution 7
def read_matrix(rows, cols):
mat = []
for _ in range(rows):
# read row numbers as floats
row = list(map(float, input().split()))
if len(row) != cols:
raise ValueError("Wrong number of columns")
mat.append(row)
return mat

# dims for A
m, p = map(int, input("Enter m p: ").split())
A = read_matrix(m, p)
# dims for B (p2 must equal p)
p2, n = map(int, input("Enter p n: ").split())
if p2 != p:
raise ValueError("Incompatible dimensions")
B = read_matrix(p, n)

# result is m x n
# initialize result matrix
C = [[0.0 for _ in range(n)] for _ in range(m)]
for i in range(m):
for j in range(n):
s = 0.0
for k in range(p):
s += A[i][k] * B[k][j] # sum product for element (i,j)
C[i][j] = s

print("Result matrix:")
for row in C:
# print formatted
print(" ".join(f"{val:.2f}" for val in row))
```

## Solution 8 — CLI Gradebook

```
# Solution 8
def compute_average(marks):    # average helper
return sum(marks) / len(marks)

def letter_grade(avg):         # letter helper
if avg >= 90: return 'A'
if avg >= 80: return 'B'
```

```
    if avg >= 70: return 'C'
    if avg >= 60: return 'D'
    return 'F'

    students = []     # list of dicts: {'name':..., 'grades':[...]}
    def add_student():
    name = input("Name: ")
    grades = []
    for i in range(3):
    g = float(input(f"Grade {i+1}: "))
    while not (0 <= g <= 100):
    g = float(input("Invalid. Enter 0-100: "))
    grades.append(g)
    students.append({'name': name, 'grades': grades})

    def show_report():
    for s in students:
    avg = compute_average(s['grades'])
    print(f"{s['name']}: Grades={s['grades']}, Avg={avg:.2f}, Grade
       ={letter_grade(avg)}")

    def class_statistics():
    if not students:
    print("No students yet"); return
    avgs = [(s['name'], compute_average(s['grades'])) for s in
       students]
    class_avg = sum(x[1] for x in avgs) / len(avgs)
    highest = max(avgs, key=lambda x: x[1])
    lowest = min(avgs, key=lambda x: x[1])
    print(f"Class avg: {class_avg:.2f}")
    print(f"Highest: {highest[0]} with {highest[1]:.2f}")
    print(f"Lowest: {lowest[0]} with {lowest[1]:.2f}")

    # main loop
    while True:
    print("\nMenu: a)Add b)Report c)Stats d)Exit")
    choice = input("Choose: ").strip().lower()
    if choice == 'a':
    add_student()
    elif choice == 'b':
    show_report()
    elif choice == 'c':
    class_statistics()
    elif choice == 'd':
    break
    else:
    print("Invalid choice")
```