CS6.201: Introduction to Software Systems

Mid Lab Exam

General Instructions

1. This exam consists of **4 questions**. Read all questions carefully before you begin.
2. **Submission filenames:** `1.sh`, `2.py`, `3.py`, `4.py` — for Questions 1 through 4 respectively.
3. **Allowed external libraries:**
    - `sys` and `math` are permitted for **all** Python questions.
    - `numpy` is permitted **only** for Question 2 (and is required for it).
    - **No other external libraries or modules may be imported** (e.g., `itertools`, `collections`, `pandas`, `openai`, etc. are all **prohibited** unless stated otherwise).
4. All Python programs must read input from **standard input** (`stdin`) and write output to **standard output** (`stdout`), unless the question explicitly specifies otherwise.
5. Ensure your scripts and programs produce the **exact** output format specified in each question.
6. Do **not** hardcode any test-case-specific values in your solutions.

Offline Documentation

A `docs/` folder is provided with the following reference material:

- `docs/numpy.pdf` — NumPy documentation
- `docs/bash.pdf` — Bash documentation
- `docs/python/index.html` — Open this file in a browser to access the Python documentation

VS Code Extension Setup

If the Python extension is not already installed, you can install it using either method:

- **Double-click** the `.vsix` file in the `extensions/` directory, **or**
- Run the following command in a terminal:

```
code --install-extension extensions/ms-python.python-2026.2.0-linux-x64.vsix
```

Reference Material

The following concepts may be useful for this exam.

Command Substitution (`$()`) — Bash Manual §3.5.4

Command substitution allows the output of a command to replace the command itself. The standard form is:

```
$(command)
```

You can capture the output of a command into a variable:

```
# Syntax
VARIABLE=$(command)

# Example
$ CURRENT_USER=$(whoami)

# Now try printing the variable
$ echo $CURRENT_USER
# Your current logged in username will be printed out
```

AWK — Extracting Columns

`awk` is a text-processing command used to analyze, filter, and manipulate structured data such as logs, CSV files, and command output. It works by scanning input line by line and performing actions based on patterns and fields.

Built-in Field Variables

AWK's built-in field variables — `$1`, `$2`, `$3`, and so on (`$0` is the entire line) — break a line of text into individual fields (columns). Fields are split on whitespace by default.

To print a specific column:

```
$ awk '{print $n}' inputfile.txt

# or you can pipe the output of another command like this
$ echo -e "hello world\nhellow orld" | awk '{print $2}'
world
orld

$ ps | awk '{print $1, $4}'
# i wonder what this prints
```

NumPy Matrix (Array) Slicing

In NumPy, slicing lets you extract parts of an array (rows, columns, or sub-matrices) using the syntax:

```
array[start:stop:step]
```

For 2D arrays, slicing is done as:

```
array[row_slice, column_slice]
```

For example, if you want the last two columns of a 3×3 matrix:

```
import numpy as np

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

print(A[:, 1:3])
# [[2 3]
#  [5 6]
#  [8 9]]
```

Note that you must still specify the row slice (`:`) to indicate that you want all available rows.

Question 1: The File Auditor

Language: Bash | **Submit as:** `1.sh`

Objective

Write a Bash script named `1.sh` that scans a specific directory for security risks. The goal is to identify **all** files owned by the current user that are potentially "leaked" — meaning other users on the system have the permission to modify or execute them.

1. Argument Handling & Validation

Your script must strictly validate the input provided by the user.

- **Count Check:** The script must take exactly **one argument** (the path to the directory).
  - If the argument is missing or if there are too many, print:
    `USAGE: ./1.sh <path to directory>`
- **Existence Check:** If the provided path does not exist on the system, print: `DIRECTORY DOES NOT EXIST`
- **Type Check:** If the path exists but is a file instead of a directory, print: `NOT A DIRECTORY`

  Note: In all error cases, the script must exit immediately after printing the message. Perform the checks in the order listed above (count → existence → type).

2. Audit Logic

For the valid directory provided, your script must find **all** files that meet **both** of the following criteria:

1. The file is **owned by the current user** running the script.
2. The file is **writable (`w`) OR executable (`x`)** by **others** (the "world" permission bit).

   Clarification: "Others" refers to the third permission group in Unix file permissions (i.e., users who are neither the file owner nor in the file's group). A file matches if the others' permission bits include `w`, `x`, or both.

   Hint: The target directory may contain hidden files (files whose names begin with `.`). These should be included in the audit.

3. Output Requirements

- The **filenames only** (not full paths) of all matching files must be saved to a file named `audit_results.txt`.
- This results file must be created **inside the target directory** (the path provided as an argument).
- The filenames must be listed **one per line**, sorted in **lexicographic order**.
- If the script is run multiple times, `audit_results.txt` must be **overwritten** with the latest results.
- The `audit_results.txt` file must be created even if no files match the criteria (i.e., it may be empty).

  Note: The `audit_results.txt` file itself should **not** appear in the audit results.

4. Technical Constraints

- **No Spaces:** You are guaranteed that filenames and directory paths will **not** contain spaces. You do not need to handle special characters or whitespace.
- **Dynamic Ownership:** Do not hardcode your username. Use system commands (e.g., `whoami`) or environment variables to identify the current user.
- **Pure Bash / Core Utilities:** You may use any standard Bash commands and constructs (loops, redirection, `ls`, `grep`, `if/else`, `find`, `awk`, `stat`, etc.).

Sample Test Case

There is only **1** sample test case for this question. A directory named `bash_test` is provided.

Successful Run

```
$ ./1.sh ./bash_test
# Result: The file ./bash_test/audit_results.txt is generated.

$ cat ./bash_test/audit_results.txt
.hid.txt
aidf.txt
akdu.txt
hnju.txt
ibcu.txt
qwet.txt
werq.txt
```

Question 2: Majat Makhanpal's Mystical Magical Matrix Multiplication Machine

Language: Python | **Required Library:** `numpy` | **Submit as:** `2.py`

Majat Makhanpal was recently hired at Moogle incorporated, where he was tasked with creating a matrix machine to perform mathematical operations for their latest AI model. Unfortunately Majat hasn't written a single line of code ever since he found out about the magical powers of Mopilot and has completely forgotten how to write code on his own. Forbidden from vibe coding and afraid of losing his job, Majat has now turned to you for help. Can you decipher his requirements and make a matrix machine capable of keeping Majat away from the fringes of unemployment?

Task

Given a matrix of size $a \times b$, perform the following operations:

Step 1 — Make the Matrix Square

If the matrix is not already square, convert it to a square matrix by adding rows (to the bottom) or columns (to the right), preserving the larger dimension:

- Let $m = \max(a, b)$.
- If $a > b$: add $(a - b)$ columns to the right.
- If $b > a$: add $(b - a)$ rows to the bottom.
- If $a = b$: skip this step.

Each added row or column must be **identical** and filled entirely with the **arithmetic mean** of all elements in the **original** (pre-padding) matrix.

Step 2 — Recursive Standardisation

Once the matrix is square (size $m \times m$), apply the following procedure recursively. The recursion operates on a "current slice" of the matrix, which initially is the entire $m \times m$ matrix. **If the current slice is 1×1, leave it unchanged and stop.**

For a current slice of size $r \times c$:

1. **Vertical split:** Divide the slice into a **left half** of size $r \times \lceil c/2 \rceil$ and a **right half** of size $r \times \lfloor c/2 \rfloor$.
2. **Standardise** all elements in the **left half** (see formula below).
3. **Horizontal split of the right half:** Divide the right half into a **top quarter** of size $\lceil r/2 \rceil \times \lfloor c/2 \rfloor$ and a **bottom quarter** of size $\lfloor r/2 \rfloor \times \lfloor c/2 \rfloor$.
4. **Standardise** all elements in the **top quarter**.
5. **Recurse** on the **bottom quarter** (it becomes the new current slice).

Step 3 — Print the Result

Print the entire resulting $m \times m$ matrix (including any padded rows/columns from Step 1).

Standardisation Formula

For an element X in a matrix slice with mean μ and standard deviation σ:

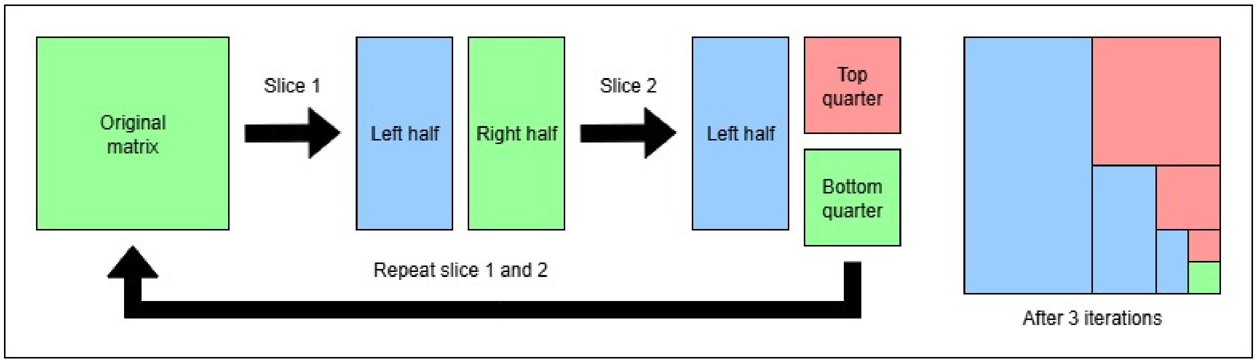

Figure 1: Illustration of the recursive splitting process

$$Z = \frac{X - \mu}{\sigma + \epsilon}$$

where $\epsilon = 10^{-8}$ is a small constant to avoid division by zero.

All intermediate computations should be performed using **floating-point arithmetic**. Only **round and convert to `int`** at the very end, when printing the final matrix.

Clarifications:

- Use **population standard deviation** (i.e., `numpy.std()` with default `ddof=0`).
- The ϵ term ensures numerical stability. If all elements in a slice are identical ($\sigma = 0$), the ϵ prevents division by zero and the result will round to 0.
- The mean used for padding in Step 1 is computed over all elements of the **original** matrix (before padding).
- "Round to nearest integer" uses standard rounding (Python's built-in `round()` or `numpy.round()`). This rounding is applied **once** to the final matrix, not during intermediate recursive steps.

Constraints

Variable	Range
$m = \max(a, b)$	$1 \leq m \leq 3000$

Expected time complexity: $O(m^2)$

Input Format

```
<rows> <cols>
<row 1 values separated by spaces>
...
<row R values separated by spaces>
```

Output Format

```
<row 1 values separated by spaces>
...
<row M values separated by spaces>
```

Elements within a row are separated by a single space. Each row ends with a newline. Output the **full** $m \times m$ matrix.

Sample Test Cases

Test Case 1

Input:

```
1 1
5
```

Output:

```
5
```

Test Case 2

Input:

```
2 2
1 2
3 4
```

Output:

```
-1 0
1 4
```

Test Case 3

Input:

```
2 3
2 4 6
8 10 12
```

Output:

```
-2 -1 -1
1 1 1
0 0 7
```

Test Case 4

Input:

```
4 1
2
```

```
2
2
2
```

Output:

```
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 2
```

Test Case 5

Input:

```
6 6
-3 1 4 1 5 9
2 6 5 3 5 8
9 7 9 3 2 3
8 4 6 2 6 4
3 3 8 3 2 7
9 5 0 -1 -2 -3
```

Output:

```
-2 -1 0 -1 0 2
-1 0 0 -1 0 1
1 1 1 -1 -1 -1
1 0 0 0 2 -1
-1 -1 1 1 0 1
1 0 -1 -1 -1 -3
```

Question 3: Substring Shenanigans

Language: Python | **Submit as:** `3.py`

Dreyas Sheb is an eccentric man with a peculiar taste in strings. He will only read a string if it satisfies a certain set of conditions. Being as considerate as he is, he reads the whole string before he discards parts of it for being "far too unpleasant to grace with his eyes". He uses this eccentricity to encode his notes to ensure no one else can study from them. Fortunately members of the *cult of in-class napping* were able to crack the code and now want you to create a program to decipher the notes.

Task

Find the **longest contiguous substring** whose first character is A_1 and last character is A_2. The substring must also contain M different specified characters, each repeated **at least** X_i times. Character counts include the first and last characters. Any characters left unspecified may appear any number of times to maximize the length of the substring.

If no such substring exists, print `-1`. Otherwise, print the substring. The input string will not contain numbers or special characters, only lowercase alphabets (`a`–`z`).

Clarifications:

- The start and end characters may be the same.
- If multiple substrings of the same maximum length satisfy all conditions, print the **leftmost** one (the one with the smallest starting index).
- A character with a frequency requirement of `0` imposes no constraint (it need not appear in the substring at all).

Constraints

Variable	Range
$\|S\|$ (length of input string)	$1 \leq \|S\| \leq 10^6$
M (number of conditional characters)	$1 \leq M \leq 26$
Characters in S	lowercase `a`–`z` only

Expected time complexity: $O(|S|)$

Input Format

```
<input string> string S
<start character> char A1 <end character> char A2
<number of conditional characters> int M
<conditional character 1> char C1 <frequency> int X1
...
<conditional character M> char CM <frequency> int XM
```

Output Format

```
-1 (if no valid substring exists)
OR
<substring> string SB
```

Sample Test Cases

Test Case 1

Input:

```
ckwvyxwezoezebwaefnwigzq
z f
3
f 1
e 3
w 0
```

Output:

```
zoezebwaef
```

Test Case 2

Input:

```
tuevuagppkyhjfwmhtywizecjwu
g h
1
m 1
```

Output:

```
gppkyhjfwmh
```

Test Case 3

Input:

```
ubvyouptunvihkbioefullz
u e
3
p 0
v 0
o 1
```

Output:

```
ubvyouptunvihkbioe
```

Test Case 4

Input:

```
lpriajrffsxyxlpgrqoqoj
a g
4
y 0
r 0
p 1
x 1
```

Output:

```
ajrffsxyxlpg
```

Test Case 5

Input:

```
moptptopqpgxplikuqbxxrtstms
p r
1
l 0
```

Output:

```
ptptopqpgxplikuqbxxr
```

Question 4: Powerset Generator

Language: Python | **Submit as:** `4.py`

I can't be bothered to write a story here, you get the point, someone is in some dire unescapable situation yada yada yada write code.

Task

Given an unsorted list of single-digit integers (0–9), generate and print its **powerset** (excluding the empty set).

Requirements

- **Duplicates:** The input may contain duplicate values. The output must contain only **distinct** sets (treat the input as a set, ignoring duplicates).
- **Ordering:**
  - Print sets in **increasing order of size** (smallest first).
  - Within each size group, print sets in **lexicographic (ascending) order**.
  - Each individual set must be **internally sorted** in increasing order.
- The input list has a maximum length of **10** elements.
- For a list with Y unique elements, the output will contain exactly $2^Y - 1$ sets.

A powerset is defined as a set of all subsets that can be generated from a set of numbers. Thus for a list with X elements and Y unique elements the powerset will contain 2^Y distinct sets including the null set. For this question you may ignore the null set, thus you are expected to print $(2^Y) - 1$ distinct sets.

Constraints

Variable	Range
X (length of input list)	$1 \leq X \leq 10$
N_i (each element)	$0 \leq N_i \leq 9$
Y (number of distinct values)	$1 \leq Y \leq 10$

Additional Constraints:

- You are **not** allowed to use `itertools`.
- The input is a list; the output is expected to be sets (i.e., no duplicate elements within a set).

Input Format

```
<space-separated integers>
```

A single line of space-separated single-digit integers.

Output Format

```
<set 1 elements separated by spaces>
```

```
<set 2 elements separated by spaces>
...
```

One set per line, elements separated by single spaces.

Sample Test Cases

Test Case 1

Input:

```
1 2 3
```

Output:

```
1
2
3
1 2
1 3
2 3
1 2 3
```

Test Case 2

Input:

```
5
```

Output:

```
5
```

Test Case 3

Input:

```
3 3 3
```

Output:

```
3
```

Test Case 4

Input:

```
1 2 2 3
```

Output:

```
1
2
3
1 2
1 3
```

```
2 3
1 2 3
```

Test Case 5

Input:

```
9 0
```

Output:

```
0
9
0 9
```