PRACTICE SHEET

What you will learn:

Memory Management & Garbage Collection, Functional Programming in Python, Module Import & Errors, Data Visualization (Matplotlib), Date & Time Handling (datetime module), Linear Algebra with NumPy, DataFrames & Pandas Method Chaining, Polynomial Operations (NumPy), Dynamic Programming, Data Cleaning (Pandas + NumPy - Processing CSVs, missing data handling)

Concept Notes:

Compare memory allocation & management in C, C++, Python, Java

Feature	**C**	**C++**	**Python**	**Java**
Allocation	Manual (malloc/free)	Manual (new/delete)	Automatic	Automatic (JVM)
Deallocation	Manual (free)	Manual (delete)	Automatic (GC)	Automatic (GC)
Safety	Unsafe (dangling pointers)	Safer (RAII)	Safe (no pointers)	Safe (no pointers)
Speed	Fast (manual control)	Fast (manual control)	Slower (GC overhead)	Medium (GC pauses)
Example	int *p = malloc(sizeof(int));	int *p = new int;	x = 10	int x = 10;

Key Points for Python:

- **Automatic Memory Management**: Uses garbage collection (GC) to clean unused objects.
- **No Pointers**: Variables are references (safer but less control).

- **Dynamic Typing**: Memory allocated at runtime (e.g., x = 10 → int object created).

Follow-Up Questions:

1. **Why does Python use garbage collection?** *(Hint: Prevents memory leaks!)*
2. **What is a "memory leak" in C/C++?** *(Manual free not called → wasted memory.)*
3. **How does Java's GC differ from Python's?** *(Java has generational GC; Python uses reference counting + cycle detector.)*

Use sys.getsizeof() to check object memory usage:

```
import sys

x = 100

print(sys.getsizeof(x))  # Output: 28 bytes (for int)
```

Demo GC: Force garbage collection with gc.collect()

Assignment: Write a Python program to create **memory-heavy objects** and monitor memory usage with tracemalloc.

[EASY] Q1) What is the output of the following code?

```
from functools import reduce

import math

result = reduce(lambda x, y: x + math.gcd(x, y), filter(lambda z: z % 2 == 0, range(10, 20)))

print(result)
```

[EASY] Q2) Consider a civil engineering dataset where bridge weights are stored. Which module combination is ideal for finding average load using functional tools?

A. map(), filter(), math
B. pandas, numpy, reduce()
C. random, math, filter()
D. datetime, reduce(), math

[EASY] Q3) **Which of the following custom modules will fail during import due to incorrect usage?**

```
# inside physics.py

def velocity(d, t):
    return d / t
```

Usage:

```
from physics import *

print(velocity(100, 0))
```

A. Will raise ZeroDivisionError
B. Will import successfully
C. Will raise ModuleNotFoundError
D. Will raise ImportError

[HARD] Q4) Generate and plot a **sinusoidal AC waveform** using Python, with the following specifications:

- Time duration: **0 to 2π seconds** (one full cycle)
- Number of samples: **100 points**
- Y-axis: **Instantaneous current (i = sin(t))**

Requirements:

1. Use **NumPy** to create the time array and compute the sine wave.
2. Use **Matplotlib** to plot the waveform with:
    - Title: **"AC Waveform"**
    - X-axis label: **"Time"**
    - Y-axis label: **"Current"**
3. Display the plot using plt.show().

Expected Output: A smooth sine wave oscillating between +1 and -1 over the 0 to 2π time period.

Checklist:
Correctly use np.linspace() for time values
Compute current using np.sin()
Properly label the plot (title, x-axis, y-axis)
Display the plot

Bonus: Add a grid to the plot using plt.grid(True).

[MEDIUM] Q5) You are given a list of project submission dates in YYYY-MM-DD format. Filter out only the projects submitted before 2023, but with these constraints:

1. Do not use the year attribute directly.
2. Instead, compare the entire date against January 1, 2023 (2023-01-01).
3. Use lambda + filter() in a single line of code.

Input:

dates = ["2022-04-05", "2023-02-10", "2021-08-12"]

Expected Output:

['2022-04-05', '2021-08-12']

Hint: Convert strings to datetime objects using strptime() and Compare dates directly (e.g., date_obj < datetime(2023, 1, 1)).

[MEDIUM] Q6) The *NumPy* module also comes with a number of built-in routines for linear algebra calculations. These can be found in the sub-module *linalg*.

linalg.det

The *linalg.det* tool computes the determinant of an array.

```
print numpy.linalg.det([[1 , 2], [2, 1]])      #Output : -3.0
```

linalg.eig

The *linalg.eig* computes the eigenvalues and right eigenvectors of a square array.

```
vals, vecs = numpy.linalg.eig([[1 , 2], [2, 1]])

print vals                                     #Output : [ 3. -1.]
```

```
print vecs                    #Output : [[ 0.70710678 -0.70710678]
                              #         [ 0.70710678  0.70710678]]
```

linalg.inv

The *linalg.inv* tool computes the (multiplicative) inverse of a matrix.

```
print numpy.linalg.inv([[1 , 2], [2, 1]])      #Output : [[-0.33333333  0.66666667]
                                                #         [ 0.66666667 -0.33333333]]
```

You are given a square matrix A with dimensions NxN. Your task is to find the determinant.
Note: Round the answer to 2 places after the decimal.

Input Format

The first line contains the integer N.
The next N lines contains the N space separated elements of array A.

Output Format

Print the determinant of A.

Sample Input

2

1.1 1.1

1.1 1.1

Sample Output

0.0

[MEDIUM] Q7) Write a solution to list the names of animals that weigh **strictly more than** 100 kilograms.

Return the animals sorted by weight in **descending order**.

The result format is in the following example.

Example 1:

Input:

DataFrame animals:

name	species	age	weight
Tatiana	Snake	98	464
Khaled	Giraffe	50	41
Alex	Leopard	6	328
Jonathan	Monkey	45	463
Stefan	Bear	100	50
Tommy	Panda	26	349

Output:

name
Tatiana
Jonathan
Tommy
Alex

DataFrame `animals`

Column Name	Type
name	object
species	object
age	int
weight	int

In Pandas, **method chaining** enables us to perform operations on a DataFrame without breaking up each operation into a separate line or creating multiple temporary variables.

Can you complete this task in just **one line** of code using method chaining?

Function Definition:

import pandas as pd

def findHeavyAnimals(animals: pd.DataFrame) -> pd.DataFrame:

Q8) You're building a **Smart Grocery Manager** app to help local kirana shop owners automate price calculations, expiry alerts, discounts, and order restocking using basic Python. Your app must process product lists with price, quantity, expiry, and discounts using **functional tools and modules**.

You’re provided a product list with each item having:

products = [

{"name": "rice", "price": 52.5, "quantity": 10, "expiry": "2025-12-01"},

{"name": "sugar", "price": 40.0, "quantity": 0, "expiry": "2023-08-10"},

{"name": "oil", "price": 150.0, "quantity": 5, "expiry": "2024-07-01"},

{"name": "milk", "price": 25.0, "quantity": 20, "expiry": "2023-07-10"},

{"name": "biscuits", "price": 10.0, "quantity": 50, "expiry": "2026-01-15"}

]

Your Tasks:

1. Use filter() with lambda to:

- Get all **in-stock items** (quantity > 0).
- Filter out **expired items** based on today's date using the datetime module.

2. Use map() to:

- Apply **10% discount** to all items whose price is above ₹50 using a lambda.

3. Use reduce() (from functools) to:

- Calculate the **total stock value** (price × quantity for each item).

4. Use math:

- Round prices **up to the nearest 5 rupees** using math.ceil().

5. Use random:

- Simulate a **restocking quantity** for out-of-stock items (random value from 10 to 50).

6. Write your own module called utilities.py that contains:

```
# utilities.py
def is_expired(date_str):
    from datetime import datetime
    today = datetime.today().date()
    return datetime.strptime(date_str, "%Y-%m-%d").date() < today
```

- Import and use this function in your main code.

7. Use from ... import to bring in reduce and your custom function.

Expected Output:

- Final list of in-stock, non-expired items with discounted, rounded prices.
- Restocked items printed with their new quantity.
- Total inventory value printed.
- Usage of all tools shown in the script.

=== Valid Inventory Items ===

Rice | Price: ₹50 | Qty: 10

Biscuits | Price: ₹10 | Qty: 50

=== Restocked Items ===

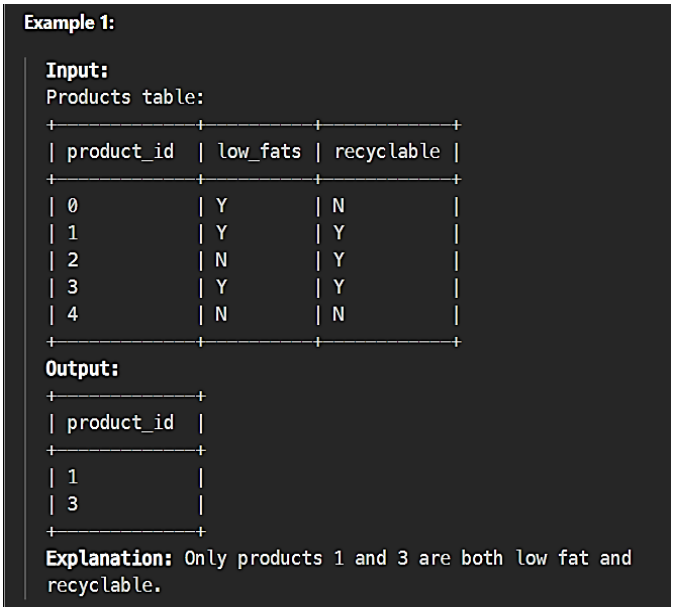
Sugar restocked from 0 to 13

Total Inventory Value: ₹1000.00

[EASY] Q9) Write a solution to find the ids of products that are both low fat and recyclable.

Return the result table in **any order**.

The result format is in the following example.



Example 1:

Input:

Products table:

product_id	low_fats	recyclable
0	Y	N
1	Y	Y
2	N	Y
3	Y	Y
4	N	N

Output:

product_id
1
3

Explanation: Only products 1 and 3 are both low fat and recyclable.

Function Definition:

import pandas as pd

def find_products(products: pd.DataFrame) -> pd.DataFrame:

[EASY] Q10) Numpy Tools:

poly

The *poly* tool returns the coefficients of a polynomial with the given sequence of roots.

```
print numpy.poly([-1, 1, 1, 10])      #Output : [  1 -11   9  11 -10]
```

roots

The *roots* tool returns the roots of a polynomial with the given coefficients.

```
print numpy.roots([1, 0, -1])         #Output : [-1.  1.]
```

polyint

The *polyint* tool returns an antiderivative (indefinite integral) of a polynomial.

```
print numpy.polyint([1, 1, 1])        #Output : [ 0.33333333  0.5         1.          0.        ]
```

polyder

The *polyder* tool returns the derivative of the specified order of a polynomial.

```
print numpy.polyder([1, 1, 1, 1])     #Output : [3 2 1]
```

polyval

The *polyval* tool evaluates the polynomial at specific value.

```
print numpy.polyval([1, -2, 0, 2], 4)  #Output : 34
```

polyfit

The *polyfit* tool fits a polynomial of a specified order to a set of data using a least-squares approach.

```
print numpy.polyfit([0,1,-1, 2, -2], [0,1,1, 4, 4], 2)

#Output : [  1.00000000e+00   0.00000000e+00  -3.97205465e-16]
```

The functions polyadd, polysub, polymul, and polydiv also handle proper addition, subtraction, multiplication, and division of polynomial coefficients, respectively.

Task

You are given the coefficients of a polynomial P.
Your task is to find the value of P at point x.

Input Format

The first line contains the space separated value of the coefficients in P.
The second line contains the value of x.

Output Format

Print the desired value.

Sample Input

1.1 2 3

0

Sample Output

3.0

IIT Dash: Visualizing Department Performance at IIT Bhilai

You are a final-year Computer Science student at IIT Bhilai working on a capstone project titled **"IIT Dash: Visual Analytics for Academic Insights"**. The goal is to help the administration visualize the **academic and placement performance** of all departments over the last 5 years.

The dashboard will help:

- Identify top-performing departments
- Spot trends in placements vs academic scores
- Visualize anomalies (e.g., sudden drop or surge)

To showcase a prototype, you've been given the following **sample dataset** for **4 departments (CSE, ECE, MECH, CIVIL)** over **5 years (2019–2023)**:

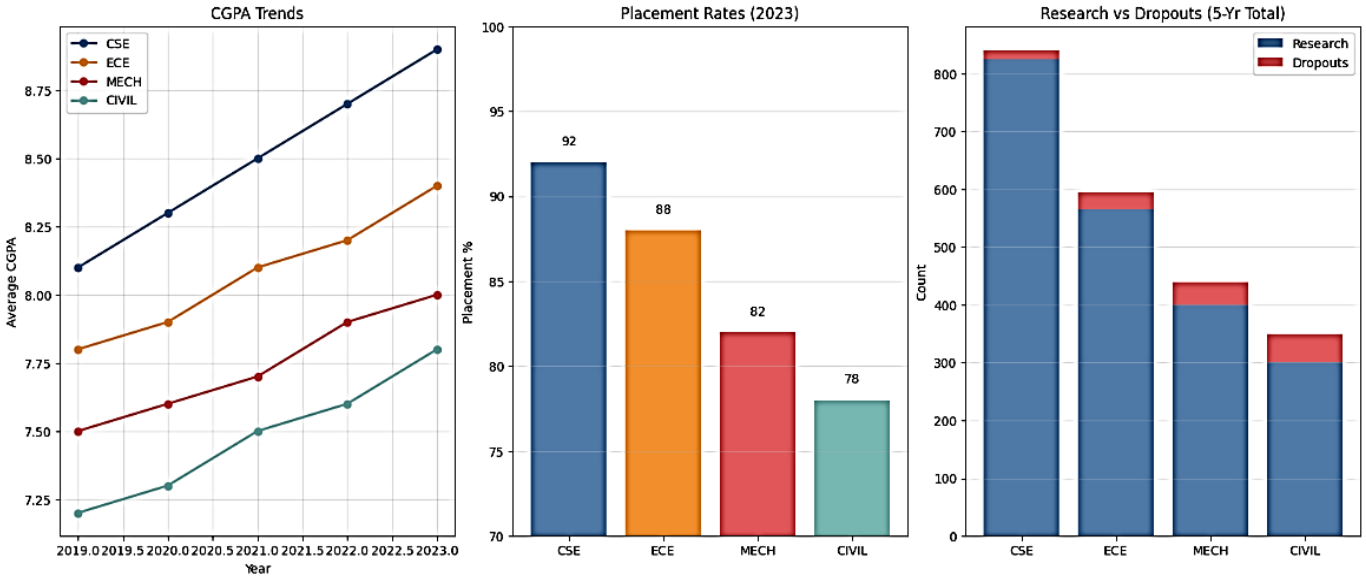
Provided Data:

- Average CGPA per department per year
- Placement Rate (%) per department per year
- Number of Research Publications
- Dropout Rates
- Student Strength

[MEDIUM] Q11) **Core Visualizations**

1. Create a 2×2 subplot grid using plt.subplots().
2. **Line Plot**: CGPA trends (2019–2023) for all departments with:
    - Distinct line styles/colors
    - Title, axis labels, legend, and grid
3. **Bar Plot**: Compare 2023 placement rates across departments.
4. **Stacked Bar Plot**: Total research publications vs dropouts per department (2019–2023).

    Output:

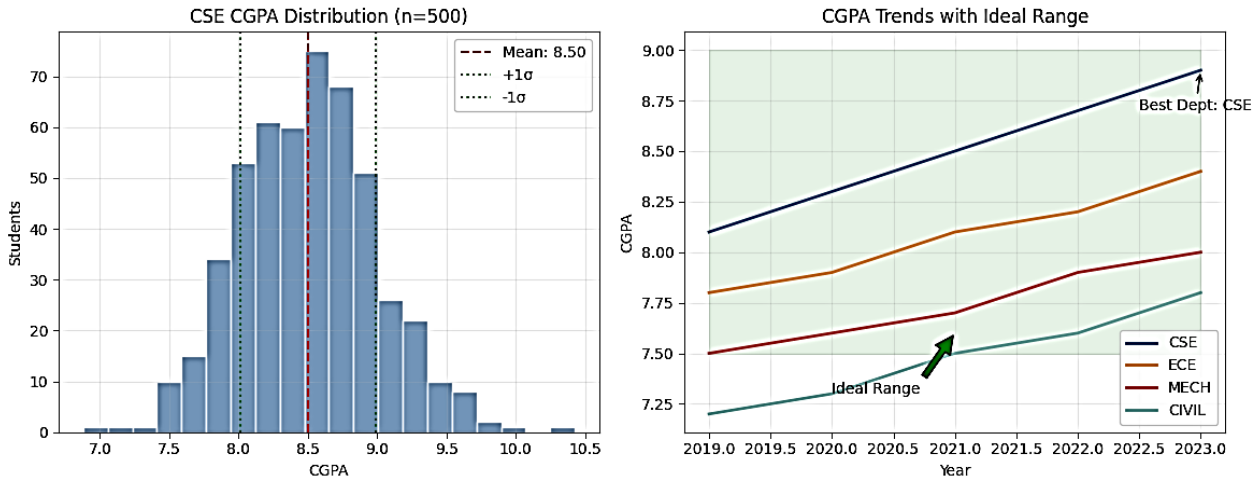


[HARD] Q12) **Advanced Features**

5. **Histogram**: Simulate 500 CSE student CGPAs using np.random.normal(). Add:
    - Mean (ax.axvline())
    - ±1 standard deviation lines
6. **Fill Between**: Highlight ideal CGPA range (7.5–9) for CSE trend line.
7. **Annotation**: Mark the best-performing department in placements.
8. **Custom Ticks**: Modify axis ticks for one plot.
9. **Global Title**: Set "IIT Department Analytics Dashboard (2019–2023)".

10. **Save Figure**: Export as "dashboard.png".

    Output:



Analyzing App User Retention & Engagement – A Data Analytics Sprint

You're a **data analyst at a tech startup** that has launched a productivity app called **FocusZen**. The product team wants to understand **user engagement, retention patterns, churn behavior**, and segment users based on their activity data. You're given a CSV of user logs and signups for analysis.

Your job is to **generate analytics reports** using **Pandas and NumPy**, and prepare derived metrics that can later be fed into a machine learning model for churn prediction.

[MEDIUM] Q13) Tasks:

Dataset provided:

1. signups.csv
   Columns: user_id, signup_date, plan_type, referral_source
2. logs.csv
   Columns: user_id, activity_date, duration_minutes, feature_used

[MEDIUM] Q13) Tasks:

1. Data Preparation

- Load both datasets with proper date parsing

- Clean data by:
    - Removing rows with missing values
    - Filtering out sessions with duration_minutes ≤ 0

2. Feature Engineering
For each user calculate:

- total_usage_time: Sum of all session durations
- active_days: Count of unique activity dates
- avg_session_duration: Mean session duration

3. Data Merging

- Combine signup and activity data using user_id
- Compute days_since_signup:

(last_activity_date - signup_date).dt.days

[HARD] Q14)

4. Retention Analysis

- Label users as:
    - "Retained": >3 active days
    - "Churned": ≤3 active days

5. User Segmentation
Categorize users into:

- "Free-Light": Free plan with <30 mins total usage
- "Free-Heavy": Free plan with ≥30 mins usage
- "Pro": All Pro users

6. Pivot Analysis
Generate:

- Average session duration by plan_type and referral_source
- User counts by plan_segment and churn_status

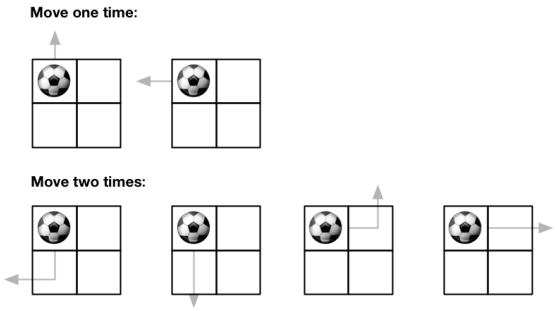
7. Output
Export final dataset to analytics_report.csv with all computed features.

[HARD] Q15) There is an m x n grid with a ball. The ball is initially at the position [startRow, startColumn]. You are allowed to move the ball to one of the four adjacent cells in the grid (possibly out of the grid crossing the grid boundary). You can apply **at most** maxMove moves to the ball.

Given the five integers m, n, maxMove, startRow, startColumn, return the number of paths to move the ball out of the grid boundary. Since the answer can be very large, return it **modulo** $10^9 + 7$.

def findPaths(self, m: int, n: int, maxMove: int, startRow: int, startColumn: int) -> int:

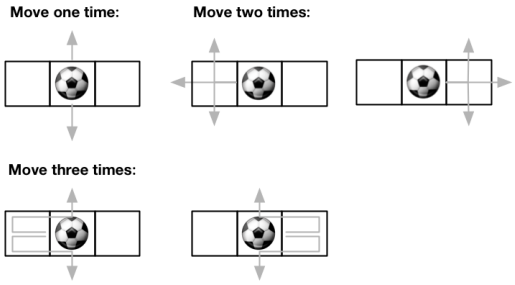
Example 1:



Input: m = 2, n = 2, maxMove = 2, startRow = 0, startColumn = 0

Output: 6

Example 2:



Input: m = 1, n = 3, maxMove = 3, startRow = 0, startColumn = 1

Output: 12