PROJECT LOG - AIRAWARE SMART AIR QUALITY PREDICTION SYSTEM

-by N.N. Sai Sreya

1.DAY 1

Topic: AirAware Smart Air Quality Prediction System

1.1 Overview:

AirAware is a smart air quality monitoring and prediction system that uses real-time pollution data and machine learning. It focuses on predicting PM2.5, CO2, NO2 levels and giving awareness about hazardous air conditions. The project includes frontend, backend, database, dashboards, and ML integration.

1.2 Objectives:

- Collect dataset containing PM2.5, CO2, NO2, etc.
- Forecast air quality using ML models such as Linear Regression, Random Forest, SVM.
- Compare Predicted vs Actual values.
- Show 3-4 months of data with analytics on a dashboard.
- Alert users if air quality becomes hazardous.
- Use Kaggle datasets or similar sources.
- Preprocess data (cleaning, normalization) and store in database.
- Build UI + API-based full-stack application.

1.3 Milestones:

1st – 25%

2nd – 50%

3rd – 75%

4th – 100%

Each milestone must include a working demo.

2. DAY 2

2.1 Team Formation:

Team 1 – N.N.Sai Sreya,Rajalaksmi, Rahul, Lokesh, Divija Nandana.

Dataset chosen: Delhi Air Quality Dataset (Kaggle).

2.2 Key Decisions:

- Build AirAware website to predict future pollution levels.
- Predict air quality for upcoming years and show trends in dashboard.
- React recommended for a beautiful UI dashboard.
- Dashboard to include last year's air quality, current quality, and predictions.
- API calls for communication between frontend and backend.
- Backend receives payload from frontend, processes it, and returns prediction results.

2.3 Additional Features:

- Graph of past few years.
- Current weather display.
- Potential integration with Google weather API.
- Accuracy visualization using heatmaps.
- Dummy sensor data input for now.
- Future enhancement: real sensors for surrounding air quality.

3. DAY 3

3.1 Topics Covered:

- Pandas, NumPy, Matplotlib
- Heatmap for model accuracy

3.2 API Concepts:

Two main Python APIs: Flask and FastAPI

- FastAPI:

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
```

- Flask API:

```
from flask_api import FlaskAPI
from flask import request
app = FlaskAPI(__name__)
@app.route("/", methods=['GET'])
```

3.3 Key Points:

- GET method sends payload in header.
- POST method sends payload in body.
- Postman is used to test backend API endpoints.
- API connects frontend and backend through requests and responses.
- Frontend payload structure must be known before writing backend.
- Changing payload breaks API functionality.
- FastAPI supports async, better performance, high concurrency.

3.4 Additional:

- WebSocket, streaming responses.
- Deployment discussions pending.

3.5 Task:

Build basic working version of AirAware dashboard with API connectivity.

4. Day 4

Topic: Git and Version Control

4.1 Commands learned:

- git add . - Adds all new/changed files to staging.
- git add <file> - Adds only the selected file to staging.
- git commit -m "msg" - Saves changes with a message.
- git push - Uploads your commits to the GitHub repo.
- git branch - Shows the branch you are currently on.
- git branch –all - Shows all branches in the project.
- git fetch –all - Gets updates from all remote branches.
- git pull - Fetches and merges latest changes from the remote branch.
- git stash - Temporarily saves uncommitted changes without pushing.
- git checkout -b <branch> - Creates and switches to a new branch.
- git checkout <branch> - Switches to an existing branch.

4.2 Virtual Environment:

- python -m venv venv
- venv/Scripts/Activate
- pip install -r requirements.txt
- deactivate

5. Day 5

Topic: Database Concepts

5.1 DB & DBMS

CRUD operations: Create, Rename, Update, Delete

5.2 Types of Databases:

Structured – MySQL, SQL, PostgreSQL

Unstructured – MongoDB

5.3 Normalization Forms:

1NF, 2NF, 3NF, BCNF, 4NF, 5NF

Concepts: Partial dependency, Transitive dependency

5.4 Keys:

Primary key, Foreign key, Composite key

5.5 Data Types:

- Master data (static)
- Transaction data (dynamic)

5.6 Frontend Pages:

- Analytical Dashboard
- About Page
- Report Page
- Login Page
- Admin Page

5.7 AI Tools:

OpenAI, Gemini, Grok, Llama, Vertex AI

5.8 ML Concepts:

- Supervised (Regression, Classification)
- Unsupervised (Clustering)
- Reinforcement learning

5.9 NLP Concepts:

- Embeddings
- Vectors
- NLTK

6. Day 6

- Project Progress Planning

6.1 Milestone Deadlines:

- 25% on 20-21 November
- 50% by 30 November
- No UI template changes after 30 November
- Final output on 1 December
- Then enhancements begin

6.2 Documentation:

- 45-page detailed project document required
- Each page should include Day and Topic
- Include Git, API, DB, Project overview, ML, and daily logs

6.3 GitHub:

- Push code to main repo

- Team repo for demo
- Personal repo for presentation
- Add teammates as contributors
- Maintain backup copy

6.4 Corporate Culture:

- Learn to communicate freely
- Understand professional workflows

7. Day 7:

7.1 Doubt Clearance & Additional Instructions

Day 7 focused mainly on clarifying students' doubts related to the AirAware project, the backend–frontend workflow, API integration, dataset usage, and Git practices. The instructor addressed questions regarding Python APIs (Flask/FastAPI), UI design guidelines, database selection, and how to structure payloads between frontend and backend.

Additional instructions were also given for the project timeline, including how to approach Milestone-1, how to organize the GitHub repository, and how each team should maintain transparency in UI development and backend logic.

Students were reminded to follow the planned structure, avoid unnecessary UI template changes after deadlines, and ensure the project was progressing toward the 25% milestone target.

On this day, we also created both the team GitHub repository and our individual personal GitHub repositories for code collaboration and documentation.

Personal Github repository: https://github.com/SaiSreya96/saisreya_personal_repo

Team Github repository: https://github.com/SaiSreya96/grp1_team_repo_infosys

8. Day 8

8.1 Preparation for Milestone-1 Presentation

On Day 8, students were given dedicated time to prepare for the Milestone-1 presentation scheduled for the next day. No new topics were taught, as the entire session was focused on getting ready for the first project review.

Since this day was fully allotted for milestone preparation, I utilized the time to work on the essential components required for the presentation. I updated and organized the **log document**, created and refined the **project PowerPoint presentation**, and made the necessary **UI changes** to improve the overall appearance and flow of the AirAware dashboard.

The main purpose of Day 8 was to ensure that the project was in a presentable and polished state for the milestone evaluation, and I worked on documentation, presentation, and UI enhancements accordingly.

9. DAY 9

Milestone 1

10.DAY 10

Milestone 1

11. Day 11

11.1. Artificial Intelligence (AI)

- AI is a set of methods that enables machines to perceive data, perform tasks, and make predictions—tasks that normally need human intelligence.

11.2. Key Subfields of AI

- ML – Machine Learning
- DL – Deep Learning
- NLP – Natural Language Processing
- RL – Reinforcement Learning
- Knowledge-based systems

11.3. Deep Learning (DL)

- DL uses **neural networks** like:
  - RNN
  - CNN

- LSTM
- Transformers

11.4. ANN — Artificial Neural Network

11.5. CNN — Convolutional Neural Network

11.6. LSTM — Long Short-Term Memory

- Has 3 gates:
  - Forget gate
  - Input gate
  - Output gate

11.7. ML Workflow

- Collect data
- Prepare & engineer features
- Train the model

12. Day 12

12.1. NLP — Natural Language Processing

- Helps machines understand, interpret, and generate human-like language.
- Main challenges: emotion, ambiguity, meaning, grammar
- Example application: Chatbots

12.2. NLTK Library

- Used for basic NLP operations
- Command: pip install nltk

12.3NLP Tasks

a). Tokenization

- Breaking text into words

- Code:
  from nltk.tokenize import word_tokenize

b). Stop Words

- Words with little meaning → removed
- Import: from nltk.corpus import stopwords

c). Stemming

- Converts words to **base/root**
- Example: playing → play
- Import: from nltk.stem import PorterStemmer

d). Lemmatization

- Also gives root word
- Example:
  - stem("better") → better
  - lemma("better") → good
- Import: from nltk.stem import WordNetLemmatizer

e). POS Tagging (Part of Speech Tagging)

- Identifies noun, verb, adjective etc.
- Uses: word_tokenize

f). Named Entity Recognition (NER)

- Extracts names, places, dates, organizations

12.4Text Preprocessing Steps

a) Lowercase

b) Remove punctuation

c) Remove numbers

d) Remove extra spaces

e) Tokenization

f) Stop word removal

g) Lemmatization / stemming

12.5 Other ML Concepts Mentioned

- Logistic Regression: used for sentiment analysis
- SVM — Support Vector Machine
- TF-IDF — Term Frequency Inverse Document Frequency
- Term Frequency (TF): how often a term appears
- Inverse Document Frequency (IDF): importance of term across documents

13. Day 13

13.1. TF–IDF

- TF (Term Frequency): Measures how often a word appears in a document.
- IDF (Inverse Document Frequency): Measures how rare a word is across all documents.
    - If a word appears in 1/1000 documents → High IDF
    - If a word appears in 999/1000 documents → Low IDF

13.2. Support Vector Machine (SVM)

- SVM comes under supervised learning.
- Mainly used for classification.
- **Margin:** Distance between the separating line and the closest data points from each class.
- Two types of margins:
    - Hard Margin**:**

▪ Perfect classification

▪ No misclassification

▪ Data is clean, no noise

- Soft Margin**:**

- For non-linear separations
- Allows some misclassification
- Boundary line is not straight

14. Day 14

14.1. RL (Reinforcement Learning)

- RL stands for Reinforcement Learning.
- Concepts:
    - Agentic AI / AI Agent
    - An agent learns by interacting with the environment and receiving rewards.
    - Multi-agent architecture: More than one agent working together.
    - Agents can operate with or without human involvement.
    - Mention of Autogen (agent framework).

14.2. CSS Flexbox

- Flexbox is a layout design method.
- Intrinsic size: Element's original size.
- Extrinsic size: Size given by the developer.

14.3. Handling Overflow (CSS)

- CSS overflow properties:
    - visible
    - hidden
    - scroll
    - auto
- overflow-x and overflow-y can be set separately.
- Minimum and maximum sizes can be controlled.

14.4. Box Sizing

- Includes content, margin, padding, border.
- Two models:
  - Content-box
  - Border-box

14.5. Layout Design

- Approaches:
  - CSS Grid
  - Flexbox
- Flex attributes include:
  - display: flex, inline-flex, grid, none
  - flex-direction

14.6. HTML & CSS Basics

- Introduction to HTML basics.
- Basic tags with small examples.

14.7. Python Integration

- Using Python with:
  - Flask API
  - FastAPI

14.8. Git & GitHub Differences

- Noted differences between Git and GitHub.

15. Day 15

15.1. Flex Wrap (CSS Flexbox)

- flex-wrap is used to adjust how items move to the next line.
- Types:
    - nowrap (default): Items stay in one line.
    - wrap**:** Items move to the next line when space is insufficient.
    - wrap-reverse: Items wrap, but in reverse order.

15.2. Flex Container Concepts

- A can be made a flex container.
- All direct children become flex items.
- A *cards container* can also be made a flex container.

Align Self (for individual items)

- flex-start
- center
- flex-end
- stretch
- auto (default)

15.3. Summary of Flex-Wrap & Align-Self

Flex-Wrap

1. nowrap
2. wrap
3. wrap-reverse

Container → align-self options

1. flex-start

2. center
3. flex-end
4. stretch
5. auto

15.4. Order Property

Used to control the order of flex items.

1. 0 (default) — comes first
2. positive values — appear later
3. negative values — come before 0's

15.5. API Key Creation (OpenAI)

Steps:

1. Search "OpenAI API key free" in Google.
2. Go to API Keys section → Generate new key.
3. Install library:
4. pip install openai
5. Import in Python:
6. from openai import OpenAI
7. Initialize client:
8. client = OpenAI()

15.6. Upcoming Tasks (as noted in your page)

- Work planned for Thursday & Friday → Milestone 2
    - Work includes using API key
- Build chatbot and Word document generator

- Push completed code to group repository
- Prepare PPT

15.7. Additional Notes

- Real-time data:
  - Need to link database with UI
  - Display real-time data
- Create API key (to be done by your monday)

15.8 Additional Notes

a)Python Chat Completion Code (OpenAI)

```
resp = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
      {"role": "system", "content": "You are a helpful assistant."},
      {"role": "user", "content": "Write a two-line poem about chai."}
    ],
    max_tokens=120
)
```

b)Explanation of Key Terms

- model → the AI model you are using.
- messages → list of messages exchanged.
- system → defines the assistant's behavior.
- user → actual prompt/question from user.
- max_tokens → controls output length.
- temperature (0.1 to 1.0) → controls creativity of response.

c) Frontend to Backend Connection (JS Fetch API)

```
let response = await fetch("http://localhost:8000/chat", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({ message: message })
});
```

Note: Used inside a <script> tag to send user message to backend API.

16. Day16

16.1. Logistic Regression

- Used for classification problems (output is Yes/No, 0/1).
- Logistic Regression falls under classification algorithms.
- It uses a linear equation:

$$z = w_1 x_1 + w_2 x_2 + \cdots + b$$

- Applies sigmoid function to convert z into probability:

$$p = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- Decision rule:
  - If p > 0.5 → class = 1
  - Else → class = 0
- Code:
- from sklearn.linear_model import LogisticRegression
- model = LogisticRegression()

16.2. Decision Tree

- Splits a big question into smaller chunks.
- Example:
    - Should I play?
        - Sunny or Rainy?
            - Rainy → Should not play
            - Sunny →
                - Hot → Should not play
                - Normal → Can play
- Uses concepts of:
    - Entropy

$$\text{Entropy} = -p\log\ p - q\log\ q$$

    - Information Gain (IG)

$$IG = \text{Entropy(parent)} - \text{Entropy(children)}$$

- Code:
- from sklearn.tree import DecisionTreeClassifier
- dt = DecisionTreeClassifier()

16.3. Random Forest

- Builds multiple decision trees, each seeing a random part of the data.
- Final output:
    - Majority vote → Classification
    - Average → Regression
- It is an ensemble model (uses many trees).
- Code:
- from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

16.5. Height Classification Example

- Given heights:
  - A = 170, B = 168 → Fit for sports
  - C = 150, D = 152 → Not fit for sports
- New heights to classify: 169, 151

(This is an example of applying ML classification based on height.)

16.6. K-Means Clustering

- Type: Unsupervised Learning
- Purpose: Group data into K clusters based on similarity

Steps:

1. Choose K centroids.
2. Assign each data point to the nearest centroid.
3. Recalculate centroids (mean of points in cluster).
4. Repeat until stable.

Objective Function:

$$\sum \text{sqrt}(x_i - \text{centroid})^2$$

Code:

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)

16.7. Linear Regression

- Predicts continuous values
  (e.g., sales, temperature, price).

- Equation:

$$y = mx + c$$

16.8. Mean Squared Error (MSE)

- Used as a loss function in regression.
- Formula:

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

16.9. Linear Regression

- Used for predicting continuous values.
- Code:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
```

16.12. XGBoost

- A powerful gradient boosting algorithm.
- It updates the model step-by-step:

$$F_{\text{new}}(x) = F_{\text{old}}(x) + \eta \cdot h(x)$$

Where:

- η (eta) = learning rate
- h(x) = a small decision tree (weak learner)

Key Points:

- Builds trees sequentially.
- Each new tree corrects the errors of the previous one.
- Adds regularization to reduce overfitting.

Code:

```
from xgboost import XGBClassifier

model = XGBClassifier()
```

17. Day 17

17.1 MySQL

SQL (Structured Query Language) is used to store, manage, and retrieve data from databases.
Basic syntax example:
SELECT * FROM table_name;

17.2 Common SQL Commands

- SELECT – retrieve data
- INSERT – add new data
- UPDATE – modify existing data
- DELETE – remove data
- CREATE – create tables/databases
- ALTER – modify table structure
- DROP – delete tables/databases
- RENAME – rename tables
- TRUNCATE – remove all rows from a table

17.3 WHERE Clause & Operators

Used to filter records.
Example: SELECT * FROM Customer WHERE country = 'Mexico';

17.4 Operators**:**

- Comparison: =, >, <, >=, <=, <> (or !=)

- Range: BETWEEN
- Pattern Matching: LIKE
- Set: IN, NOT IN
- Logical: AND, OR, NOT
- Sorting: ORDER BY

17.5 LIKE Pattern Matching

- a% → starts with a
- %a → ends with a
- %a% → contains a
- _a% → second letter is a
- a_% → starts with a, at least 2 characters
- a__% → starts with a, at least 3 characters
- a%o → starts with a and ends with o

17.6 Aggregate Functions

- MIN() – smallest value
- MAX() – largest value
- COUNT() – total number of rows
- AVG() – average value

17.7 Aliases

Used to rename columns or tables temporarily.
Example: SELECT name AS student_name FROM student;

17.8 Joins

Used to combine data from multiple tables.

17.9 Types of Joins:

- INNER JOIN – matching rows from both tables
- LEFT JOIN – all rows from left table + matched rows from right
- RIGHT JOIN – all rows from right table + matched rows from left
- FULL JOIN (if supported) – all rows from both tables
- CROSS JOIN – cartesian product

Example tables:
Table 1: (student) → std_id, std_name, address
Table 2: (subject) → sbj_id, stud_id, sbj_name

17.10 UNION & UNION ALL

- UNION – combines result sets, removes duplicates
- UNION ALL – combines result sets, keeps duplicates

17.11 GROUP BY & HAVING

- GROUP BY – groups rows based on a column
- HAVING – applies conditions on grouped data (used with aggregates)

18. DAY 18

Milestone -2 preparation

19. DAY 19

Milestone -2

20. DAY 20

Milestone -2

21. DAY 21

- Presenting team UI
- python build in methods:

1. lower(): Converts the string to lowercase
2. upper(): Converts the string to uppercase
3. split(): Splits the string into a list (default: by spaces).
4. strip(): Removes leading and trailing spaces (or given characters).
5. join(): Joins elements of a list into a string using a separator.
6. replace(): Replaces part of a string with another string.
7. startswith(): Checks if the string starts with a given value → returns True/False.
8. endswith(): Checks if the string ends with a given value → returns True/False.
9. find(): Returns the index of a substring (or -1 if not found).
10. isdigit(): Returns True if all characters are digits.
11. isalpha(): Returns True if all characters are alphabetic.

22. DAY 22

22.1 LIST METHODS

1. append() – adds one item at the end

```
x = [1,2]

x.append(3)   # [1,2,3]
```

2. extend() – adds multiple items

```
x = [1,2]

x.extend([3,4])   # [1,2,3,4]
```

3. insert() – adds item at a specific position

```
x = [1,3]

x.insert(1,2)   # [1,2,3]
```

4. remove() – removes first matching item

```
x = [1,2,3]

x.remove(2)   # [1,3]
```

5. pop() – removes and returns item (default last)

```
x = [1,2,3]

x.pop()     # returns 3 → [1,2]
```

6. index() – returns position of item

```
x = [10,20,30]

x.index(20)   # 1
```

7. count() – counts occurrences

```
x = [1,2,1,1]

x.count(1)   # 3
```

8. sort() – sorts list

```
x = [3,1,2]

x.sort()   # [1,2,3]
```

9. reverse() – reverses list order

```
x = [1,2,3]

x.reverse()   # [3,2,1]
```

22.2 **DICTIONARY METHODS**

1. get() – returns value safely

```
d = {"a":1}

d.get("a")   # 1
```

2. keys() – returns all keys

```
d = {"a":1}

d.keys()    # dict_keys(['a'])
```

3. values() – returns all values

```
d = {"a":1}

d.values()   # dict_values([1])
```

4. items() – returns key–value pairs

```
d = {"a":1}

d.items()   # dict_items([('a',1)])
```

5. update() – adds/changes items

d = {"a":1}

d.update({"b":2}) # {"a":1,"b":2}

6. pop() – removes item by key

d = {"a":1,"b":2}

d.pop("a") # returns 1 → {"b":2}

7. popitem() – removes last inserted pair

d = {"a":1,"b":2}

d.popitem() # removes ("b",2)

8. clear() – removes all items

d = {"a":1}

d.clear() # {}

22.3 SET METHODS

1. add() – adds an element

s = {1,2}

s.add(3) # {1,2,3}

2. remove() – removes item (error if missing)

s = {1,2,3}

s.remove(2) # {1,3}

3. discard() – removes item (no error)

s = {1,2,3}

s.discard(5) # safe

4. pop() – removes random item

s = {1,2,3}

s.pop() # removes any one item

5. clear() – empties set

```
s = {1,2}

s.clear()   # set()
```

6. union() – combines sets

```
a = {"a","b","c"}

b = {"c","d","e"}

a.union(b)   # {'a','b','c','d','e'}
```

7. intersection() – common elements

```
a = {"a","b","c"}

b = {"c","d","e"}

a.intersection(b)   # {'c'}
```

8. difference() – items in A not in B

```
a = {"a","b","c"}

b = {"c","d","e"}

a.difference(b)   # {'a','b'}

b.difference(a)   # {'d','e'}
```

22.4 FILE I/O METHODS

1. open() – opens a file

```
f = open("a.txt","r")
```

2. read() – reads whole file

```
data = f.read()
```

3. readline() – reads one line

```
line = f.readline()
```

4. readlines() – reads all lines as list

```
lines = f.readlines()
```

5. write() – writes a string

```
f = open("a.txt","w")
f.write("hello")
```

6. writelines() – writes list of lines

```
f.writelines(["a\n","b\n"])
```

7. close() – closes file

```
f.close()
```

22.5 GENERAL PURPOSE FUNCTIONS

1. len() – gives length

```
len([1,2,3])   # 3
```

2. range() – generates numbers

```
list(range(3))   # [0,1,2]
```

3. print() – prints output

```
print("hello")
```

4. type() – shows data type

```
type(10)   # int
```

5. id() – memory address

```
id(5)
```

6. sorted() – returns sorted list

```
sorted([3,1,2])   # [1,2,3]
```

7. enumerate() – gives index + value

```
list(enumerate(['a','b']))
# [(0,'a'), (1,'b')]
```

8. zip() – pairs elements

```
list(zip([1,2], ['a','b']))
# [(1,'a'), (2,'b')]
```

23. DAY 23

23.1. Conversion Functions

int() – converts to integer

```
x = int("10")
print(x)    # 10
```

float() – converts to float

```
x = float("3.14")
print(x)    # 3.14
```

str() – converts to string

```
x = str(123)
print(x)    # "123"
```

list() – converts to list

```
x = list("abc")
print(x)    # ['a', 'b', 'c']
```

tuple() – converts to tuple

```
x = tuple([1,2,3])
print(x)    # (1, 2, 3)
```

set() – converts to set (removes duplicates)

```
x = set([1,2,2,3])
print(x)    # {1, 2, 3}
```

dict() – creates dictionary

```
x = dict(a=1, b=2)
print(x)    # {'a': 1, 'b': 2}
```

23.2. Mathematical Functions

abs() – absolute value

print(abs(-5)) # 5

sum() – sum of iterable

print(sum([1,2,3])) # 6

min() & max() – smallest/largest value

print(min(3,1,4)) # 1

print(max(3,1,4)) # 4

pow() – power

print(pow(2,3)) # 8

round() – rounds number

print(round(3.567, 2)) # 3.57

23.3. Functional Programming Tools

filter() – filters elements

nums = [1,2,3,4,5]

evens = list(filter(lambda x: x%2==0, nums))

print(evens) # [2, 4]

map() – applies function to all elements

nums = [1,2,3]

squares = list(map(lambda x: x*x, nums))

print(squares) # [1, 4, 9]

reduce() (from functools)

from functools import reduce

result = reduce(lambda a,b: a+b, [1,2,3,4])

print(result) # 10

Lambda Function – anonymous function

mul = lambda a, b: a * b

```
print(mul(3,4))     # 12
```

23.4. Input & Output

input()

```
name = input("Enter name: ")
print(name)
```

format()

```
msg = "Name: {}, Age: {}".format("Sreya", 20)
print(msg)
```

23.5. Class & Object Related

getattr() – get attribute

```
class A:
   x = 10

a = A()
print(getattr(a, "x"))    # 10
```

setattr() – set attribute

```
setattr(a, "y", 20)
print(a.y)   # 20
```

hasattr() – check attribute

```
print(hasattr(a, "x"))   # True
```

delattr() – delete attribute

```
delattr(a, "x")
```

isinstance() – check object type

```
print(isinstance(5, int))   # True
```

issubclass() – check subclass

```
class B(A):
    pass


print(issubclass(B, A))  # True
```

23.6. Miscellaneous

globals() – global variables

```
print(globals().keys())
```

locals() – local variables

```
def f():
    a = 10
    print(locals())
f()
```

callable() – checks if object is callable

```
print(callable(print))   # True
```

eval() – executes string expression

```
print(eval("2+3"))   # 5
```

exec() – executes Python code

```
exec("x = 10; print(x)")
```

23.7. Exception Handling

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

```
finally:
    print("This always runs")
```

23.8. Memory & Object Management

del keyword – deletes variable

```
x = 5
del x
```

gc.collect() – forces garbage collection

```
import gc
gc.collect()
```

23.9. Working With Iterables

iter() – creates iterator

```
it = iter([1,2,3])
```

next() – get next element

```
print(next(it))   # 1
```

23.10. Decorators & Metaprogramming

@staticmethod

```
class A:
    @staticmethod
    def show():
        print("Static method")

A.show()
```

@classmethod

```
class A:
    @classmethod
    def demo(cls):
        print("Class method")

A.demo()
```

23.11. Context Managers

with / as

```
with open("file.txt", "w") as f:
    f.write("Hello")
```

23.12. import Statement

```
import math
print(math.sqrt(16))
```

24 DAY 24

Python Interview Concepts Summary

24.1 Python Key Features

Python provides simplicity, readability, portability, rich libraries, dynamic typing, and automatic memory management.

24.2 Python Data Types

Core types include: **int, float, complex, str, list, tuple, set, dict, bool, NoneType**.

24.3 PEP 8

PEP 8 is Python's official style guide that ensures **clean, readable, and consistent** coding practices.

24.4 Mutable vs Immutable

Mutable objects can change after creation (list, dict, set). Immutable objects cannot change (int, float, str, tuple).

24.5 Built-in Features

Includes built-in functions, extensive standard library, exception handling, dynamic typing, and strong community support.

24.6 Python Memory Management

Python uses **reference counting** and an automatic **garbage collector** to manage memory.

24.7 Indentation

Python uses indentation instead of braces to define code blocks, ensuring readability.

24.8 Python as an Interpreted Language

Python code is first **compiled to bytecode**, then executed by the **Python Virtual Machine (PVM)**.

24.9 Namespaces

A namespace maps **names to objects**, helping avoid naming conflicts (local, global, built-in).

24.10 List vs Tuple

Lists are **mutable**, slower, allow updates.
Tuples are **immutable**, faster, and used for fixed data.

24.11 Sets Usage

Sets store **unique**, unordered elements and support operations like union, intersection, and difference.

24.12 Dictionaries

Dictionaries store **key–value pairs** and allow fast lookups and updates.

24.13 Merging Dictionaries

Use:

dict1.update(dict2)

24.14 Removing Duplicates from a List

Convert to a set or use list comprehension/filtering.

24.15 Flattened Nested List

Converting a list with inner lists into a **single-level** list.

24.16 Shallow Copy vs Deep Copy

Shallow copy copies references; deep copy copies actual nested objects.

24.17 List Slicing

Slicing allows extracting parts of a list using:
list[start: end: step].

24.18 Reversing a List

Use slicing:

lst[::-1]

24.19 Frozen Set

An **immutable** version of a set, useful as dictionary keys or constants.

24.20 Difference Between is and ==

- == checks **value equality**
- is checks **identity (same object in memory)**

24.21 Python Code for Stack and Queue

Stack (LIFO):

```
stack = []

stack.append(10)

stack.pop()
```

Queue (FIFO):

```
from collections import deque

q = deque()

q.append(10)

q.popleft()
```

DAY 25

PYTHON INTERVIEW QUESTIONS CONTINUATION

25.1 Difference Between Functions and Methods

Functions are independent blocks of code; methods are functions defined inside a class and accessed through objects.

**25.2 *args and kwargs

*args accepts variable-length positional arguments; **kwargs accepts variable-length keyword arguments.

25.3 Recursion Example

A function calling itself, e.g., factorial or Fibonacci implementation.

25.4 Decorators

Functions that modify other functions' behavior without changing their original code.

25.5 Generators

Functions using yield to produce values one at a time, saving memory.

25.6 Iterator vs Generator

Iterators use __iter__() and __next__(); generators automatically create these using yield.

25.7 Static Method and Class Method

Static methods don’t use class/object; class methods receive the class (cls) as the first argument.

25.8 Magic Methods

Special methods like __init__, __str__, __len__, enabling operator overloading and object behavior customization.

25.9 Monkey Patching

Dynamically modifying or replacing methods/functions at runtime.

25.10 Error vs Exception

Errors are serious issues (syntax, logical) while exceptions are runtime issues handled using try–except.

25.11 Custom Exception / Custom Error

User-defined exceptions created by inheriting from Exception class.

25.12 Exception Chaining

Linking exceptions using raise new_exception from original_exception.

25.13 Module vs Package

A **module** is a single Python .py file; a **package** is a folder containing multiple modules with __init__.py.

25.14 Python Virtual Environment Creation

python -m venv env

Used to isolate project dependencies.

25.15 Difference Between tell() and seek()

tell() returns the current file pointer position; seek() moves the file pointer to a specific position.

25.16 os.path.exists()

Used to check if a file or directory exists.

25.17 Absolute vs Relative Paths

Absolute path: full path from root.
Relative path: path relative to current working directory.

25.18 shutil.copy() and shutil.move()

copy() duplicates a file; move() transfers a file to a new location.

25.19 Pivot in Pandas

Reshapes data by converting column values into headers.

25.20 GroupBy in Pandas

Groups data based on column values and applies aggregation functions.

25.21 Concat vs Append

concat() combines multiple DataFrames; append() adds rows (append is deprecated).

25.22 What is GIL?

Global Interpreter Lock—a mechanism that allows only one thread to execute Python bytecode at a time.

25.23 What is a Metaclass?

A class that defines how other classes are created; “class of a class.”

25.24 Multithreading

Running multiple threads concurrently—used for I/O-bound tasks.

25.25 Multiprocessing

Running multiple processes—used for CPU-bound tasks to bypass GIL.

25.26 Feature Scaling

Normalizing or standardizing numerical data to improve machine-learning model performance.

25.27 Splitting Dataset into Train & Test

Using sklearn:

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2)
```

25.28 Features of Java

Java offers platform independence, OOP concepts, robustness, security, portability, multithreading, and automatic memory management.

25.29 JVM, JRE, JDK (and Differences)

- **JVM**: Executes Java bytecode.
- **JRE**: JVM + libraries needed to run Java programs.
- **JDK**: JRE + development tools (compiler, debugger). JDK > JRE > JVM.

25.30 Access Modifiers in Java

public, private, protected, and **default** determine visibility of classes and members.

25.31 Difference Between == and .equals()

== checks reference equality; .equals() checks value/content equality.

25.32 What is a Constructor?

Special method used to initialize objects when they are created.

25.33 Can Constructors Be Private?

Yes, used in **Singleton patterns** or to restrict object creation.

25.34 Overloading vs Overriding

- **Overloading**: Same method name, different parameters (compile-time).
- **Overriding**: Same method, same signature in subclass (runtime).

25.35 Primitive Data Types

byte, short, int, long, float, double, char, boolean.

25.36 Type Casting

Converting one data type into another—implicit or explicit.

25.37 Static vs Non-static Variables

Static: belongs to class, shared by all objects.
Non-static: unique to each object.

25.38 final vs finally vs finalize

- **final**: keyword for constants, preventing override or inheritance.
- **finally**: block in exception handling that always executes.
- **finalize()**: method called by garbage collector before object destruction.

25.39 Use of this and super

this refers to current object; super refers to parent class.

25.40 String vs StringBuffer vs StringBuilder

String: immutable.
StringBuffer: mutable, thread-safe.
StringBuilder: mutable, not thread-safe, faster.

25.41 Array vs ArrayList

Array: fixed size, stores same data type.
ArrayList: dynamic size, part of Collections Framework.

25.42 What is Collections Framework?

A set of classes and interfaces (List, Set, Map, Queue) for storing and manipulating data.

25.43 List vs Set

List: ordered, allows duplicates.
Set: unordered, no duplicates.

25.44 ArrayList vs LinkedList

ArrayList: fast for access, slow for insert/delete.
LinkedList: fast insert/delete, slower access.

25.45 ConcurrentHashMap

Thread-safe version of HashMap, uses segmentation for better concurrency.

25.46 Fail-fast vs Fail-safe Iterators

Fail-fast: throw error if collection modified during iteration (e.g., ArrayList).
Fail-safe: operate on copy, no error (e.g., ConcurrentHashMap).

25.47 Can We Have Multiple Catch Blocks?

Yes, multiple catch blocks handle different exception types.

25.48 Difference Between throw and throws

throw is used to explicitly throw an exception.
throws declares exceptions a method might throw.

25.49 Process vs Thread

Process: independent program with its own memory.
Thread: lightweight subprocess sharing memory.

25.50 What is Synchronization?

Process of controlling access to shared resources to avoid conflicts in multithreading.

25.51 sleep() vs wait()

sleep(): pauses thread for given time without releasing lock.
wait(): releases lock and waits for notification.

25.52 Deadlock

Two or more threads waiting indefinitely for each other's locks.

25.53 What is Stream API?

Used for functional-style operations on collections like map, filter, reduce.

25.54 Serialization

Converting an object to a byte stream for storage or transfer.

25.55 Inner Class

Classes defined inside another class—types: static, member, local, anonymous.

25.56 Compile-time vs Runtime Polymorphism

Compile-time: method overloading.
Runtime: method overriding.

25.57 Method Hiding

When a subclass defines a static method with the same name as parent's static method—does not override, only hides.

26. DAY 26

Preparation for milestone -3 and project related work

27. DAY 25 - Software Development Life Cycle (SDLC)

27.1 SDLC Roles

- Functional Consultant / Business Analyst
- Developer / Programmer
- Testing / QA Team
- Administration Team
- Product / Project Management
- Sales & Marketing
- Human Resource
- Documentation Team
- Support Team

27.2 Documents

- Functional Specification Document (FSD)
- Technical Specification Document (TSD)
- Test Script
- Test Result

27.3 Development Process

- Requirement gathering
- Prototype creation
- Development
- Unit Testing
- Quality Testing
- Production deployment

27.4 Servers

- Development Server
- Quality (QA) Server
- Production Server

27.5 Testing / QA

- Test scripts prepared based on FSD
- Testing performed in Quality server
- Test results generated

27.6 Administration

- Network management
- Database management
- Application management

27.7 Waterfall Model

- Linear and sequential development model
- Each phase must be completed before the next starts
- Phases:
    - Requirement Analysis

    - System Design
    - Implementation (Coding)
    - Testing
    - Deployment
    - Maintenance
- Suitable for small and well-defined projects
- Changes are difficult once a phase is completed

27.8 Agile Methodology

- Iterative and incremental development model
- Project divided into small units called **sprints**
- Continuous customer feedback
- Flexible to changes
- Frequent testing and delivery
- Better suited for large and evolving projects

27.9 Agile Roles

- Product Owner
- Scrum Master
- Development Team

27.10 Agile Practices

- Daily stand-up meetings
- Sprint planning
- Sprint review
- Sprint retrospective

28. DAY 28

* preparation of project for milestone 3

29. DAY 29

29.1 A* Algorithm (Path Finding Algorithm)

A* algorithm is a **path-finding algorithm** used to find the **shortest path** between a **start node** and a **goal node** in a weighted graph.

Example:

- **Start Node:** A
- **Goal Node:** F
- **Nodes:** A, B, C, D, E, F

Path followed:

- [A]
- [A, B]
- [A, B, D]
- [A, B, D, F]

Shortest Path: A → B → D → F
Total Cost: 5

29.2 Bubble Sort

Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order.

Example:
Array: 5, 3, 1

Pass 1 → 3, 1, 5
Pass 2 → 1, 3, 5

Sorted Array: 1, 3, 5

29.3 Selection Sort

Selection Sort selects the **smallest element** from the unsorted part and places it at the beginning.

Example:
Array: 64, 25, 12

- Smallest = 12 → swap with 64
- Array becomes: 12, 25, 64

Sorted Array: 12, 25, 64

29.4 Insertion Sort

Insertion Sort builds the sorted array **one element at a time** by inserting elements into their correct position.

Technique:

1. Assume first element is sorted
2. Pick the next element
3. Compare with previous elements
4. Shift larger elements right
5. Insert element in correct position

Example:
Array: 8, 3, 5

- Insert 3 → 3, 8, 5
- Insert 5 → 3, 5, 8

Sorted Array: 3, 5, 8

29.5 Merge Sort

Merge Sort uses **divide and conquer** approach by dividing the array and then merging sorted parts.

Example:
Array: 6, 3, 9, 5

- Divide → [6, 3], [9, 5]
- Sort → [3, 6], [5, 9]
- Merge → [3, 5, 6, 9]

Sorted Array: 3, 5, 6, 9

29.6 Quick Sort

Quick Sort selects a **pivot element**, partitions the array, and sorts recursively.

Example:
Array: 10, 7, 8, 9

- Pivot = 8
- Left → 7
- Right → 10, 9

After sorting → 7, 8, 9, 10

29.7 Heap Sort

Heap Sort uses a **binary heap** data structure to sort elements.

Example:
Array: 4, 10, 3

- Build Max Heap → 10, 4, 3
- Remove max → 4, 3

Sorted Array: 3, 4, 10

30. DAY 30

MILESTONE-3

31. DAY 31

MILESTONE-3

32. DAY 32

PROJECT PREPARATION