this is AI software engineer role technical test examples (python)

Flashcard 1: Detect Cycle in a Directed Graph (DFS)

📌 **Problem:** Given a directed graph, detect if it contains a cycle.
🔹 **Concepts:** Graph traversal, Depth-First Search (DFS), cycle detection.

```
from collections import defaultdict

def has_cycle(graph):
    visited = set()
    rec_stack = set()

    def dfs(node):
        if node in rec_stack:
            return True  # Cycle detected
        if node in visited:
            return False  # Already processed

        visited.add(node)
        rec_stack.add(node)

        for neighbor in graph[node]:
            if dfs(neighbor):
                return True

        rec_stack.remove(node)
        return False

    for node in graph:
        if node not in visited:
            if dfs(node):
                return True
    return False

# Example usage
graph = {0: [1], 1: [2], 2: [0]}  # Contains a cycle
print(has_cycle(graph))  # Output: True
```

Flashcard 2: Find the Most Frequent Element in an Array

📌 **Problem:** Given a list, return the element that appears most frequently.
🔹 **Concepts:** Hash tables (dictionary), Counter from collections.

```
from collections import Counter

def most_frequent(nums):
    count = Counter(nums)
    return max(count, key=count.get)

print(most_frequent([1, 3, 3, 2, 1, 3, 2, 3]))  # Output: 3
```

Flashcard 3: Preprocess Data for Machine Learning

📌 **Problem:** Implement a function to preprocess a dataset.
🔹 **Concepts:** Pandas, handling missing values, feature scaling.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

def preprocess_data(df):
    df.fillna(df.mean(), inplace=True)  # Fill missing values with
mean
    scaler = StandardScaler()
    df.iloc[:, :-1] = scaler.fit_transform(df.iloc[:, :-1])  # Scale
features
    return df

# Example usage
data = {'Feature1': [1, 2, None, 4], 'Feature2': [10, None, 30, 40],
'Label': [0, 1, 0, 1]}
df = pd.DataFrame(data)
print(preprocess_data(df))
```

Flashcard 4: REST API with Flask

📌 **Problem:** Create a simple Flask API for a bookstore.
🔹 **Concepts:** Flask, CRUD operations.

```
from flask import Flask, jsonify, request

app = Flask(__name__)

books = []

@app.route("/books", methods=["GET"])
def get_books():
    return jsonify(books)

@app.route("/books", methods=["POST"])
def add_book():
    book = request.json
    books.append(book)
    return jsonify(book), 201

@app.route("/books/<int:index>", methods=["DELETE"])
def delete_book(index):
    if 0 <= index < len(books):
        removed = books.pop(index)
        return jsonify(removed)
    return jsonify({"error": "Invalid index"}), 404

if __name__ == "__main__":
    app.run(debug=True)
```

Flashcard 5: AI Fraud Detection (Fix the Bug)

📌 **Problem:** Fix a bug in a fraud detection function.
🔹 **Concepts:** Pandas, logical operators.

```
import pandas as pd

def detect_fraud(transactions):
    transactions['is_fraud'] = transactions['amount'] >= 1000  #
Fixed: '>=' instead of '>'
    return transactions[['transaction_id', 'is_fraud']]

# Example usage
df = pd.DataFrame({'transaction_id': [1, 2, 3], 'amount': [500,
1500, 1000]})
print(detect_fraud(df))
# Expected: Transactions with 1500 and 1000 should be marked as
fraud.
```

Flashcard 6: Merge Two Sorted Lists

📌 **Problem:** Merge two sorted linked lists into one sorted list.
🔹 **Concepts:** Linked lists, recursion.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def merge_sorted_lists(l1, l2):
    if not l1 or not l2:
        return l1 or l2
    if l1.val < l2.val:
        l1.next = merge_sorted_lists(l1.next, l2)
        return l1
    else:
        l2.next = merge_sorted_lists(l1, l2.next)
        return l2
```

Flashcard 1: Find First Non-Repeating Character

📌 **Problem:** Given a string, find the first non-repeating character.
🔹 **Concepts:** Hash table (dictionary), string traversal.

```
def first_unique_char(s):
    count = {}
    for char in s:
        count[char] = count.get(char, 0) + 1
    for char in s:
        if count[char] == 1:
            return char
    return None  # No unique character found

print(first_unique_char("aabccdeff"))  # Output: 'b'
```

Flashcard 2: Dijkstra's Algorithm (Shortest Path)

📌 **Problem:** Find the shortest path in a weighted graph.
🔹 **Concepts:** Graphs, priority queue (heap).

```
import heapq

def dijkstra(graph, start):
    heap = [(0, start)]  # (cost, node)
    shortest = {start: 0}

    while heap:
        cost, node = heapq.heappop(heap)

        for neighbor, weight in graph.get(node, {}).items():
            new_cost = cost + weight
            if neighbor not in shortest or new_cost <
shortest[neighbor]:
                shortest[neighbor] = new_cost
                heapq.heappush(heap, (new_cost, neighbor))

    return shortest

graph = {'A': {'B': 1, 'C': 4}, 'B': {'C': 2, 'D': 5}, 'C': {'D':
1}, 'D': {}}
print(dijkstra(graph, 'A'))
```

```
# Output: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
```

Flashcard 3: Train a Decision Tree Model

📌 **Problem:** Train a model on a dataset using `DecisionTreeClassifier`.
🔹 **Concepts:** Scikit-learn, model training.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Dummy dataset
data = {'Feature1': [1, 2, 3, 4, 5], 'Feature2': [2, 3, 4, 5, 6],
'Label': [0, 1, 0, 1, 0]}
df = pd.DataFrame(data)

X = df[['Feature1', 'Feature2']]
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))  # Example
Output: 1.0
```

Flashcard 4: Extract Emails from a Text File

📌 **Problem:** Extract all emails from a file.
🔹 **Concepts:** Regex, file handling.

```
import re

def extract_emails(file_path):
    with open(file_path, 'r') as file:
        content = file.read()
    return
re.findall(r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}',
content)

# Example usage
emails = extract_emails("sample.txt")
print(emails)  # Output: ['example@email.com', 'test@domain.org']
```

Flashcard 5: Fix a Bug in Fraud Detection Code

📌 **Problem:** Fix a bug in a function that classifies transactions as fraudulent.
🔹 **Concepts:** Debugging, pandas.

```
import pandas as pd

def detect_fraud(transactions):
    transactions['is_fraud'] = transactions['amount'] > 1000  # Bug:
'>=' should be used?
    return transactions[['transaction_id', 'is_fraud']]

# Sample data
data = {'transaction_id': [1, 2, 3], 'amount': [500, 1500, 1000]}
df = pd.DataFrame(data)

print(detect_fraud(df))
# Expected: Transaction with 1000 should also be marked as fraud
```

Flashcard 7: Sentiment Analysis with TextBlob

📌 **Problem:** Implement a function that classifies text sentiment as **positive, negative, or neutral**.
🔹 **Concepts:** NLP, Sentiment Analysis, TextBlob.

```
from textblob import TextBlob

def classify_sentiment(text):
    polarity = TextBlob(text).sentiment.polarity
    if polarity > 0:
        return "Positive"
    elif polarity < 0:
        return "Negative"
    else:
        return "Neutral"

# Example usage
print(classify_sentiment("I love this product!"))  # Output:
Positive
print(classify_sentiment("This is the worst service ever."))  #
Output: Negative
```

Flashcard 8: Find Missing Number in an Array

📌 **Problem:** Given an array of **n-1** unique numbers from **1 to n**, find the missing number.
🔹 **Concepts:** Summation formula, XOR approach.

```
def find_missing_number(arr, n):
    expected_sum = n * (n + 1) // 2
    return expected_sum - sum(arr)

# Example usage
arr = [1, 2, 4, 5, 6]  # Missing number is 3
print(find_missing_number(arr, 6))  # Output: 3
```

Flashcard 9: Reverse a Linked List

📌 **Problem:** Reverse a given linked list.
🔹 **Concepts:** Linked List, Iteration.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverse_linked_list(head):
    prev, curr = None, head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    return prev
```

Flashcard 10: K-Nearest Neighbors Algorithm (KNN) from Scratch

📌 **Problem:** Implement a simple KNN classifier.
🔹 **Concepts:** Machine Learning, Euclidean Distance, Classification.

```
import numpy as np
from collections import Counter

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((np.array(x1) - np.array(x2)) ** 2))

def knn_predict(X_train, y_train, x_test, k=3):
    distances = [(euclidean_distance(x_train, x_test), y) for
x_train, y in zip(X_train, y_train)]
    nearest_neighbors = sorted(distances)[:k]
    labels = [label for _, label in nearest_neighbors]
    return Counter(labels).most_common(1)[0][0]

# Example usage
X_train = [[2, 3], [5, 4], [3, 7], [8, 8]]
y_train = ["A", "B", "A", "B"]
x_test = [4, 5]
```

```
print(knn_predict(X_train, y_train, x_test))  # Expected output: "A"
or "B" depending on neighbors
```

Flashcard 11: Implement a Rate Limiter (Token Bucket Algorithm)

📌 **Problem:** Design a rate limiter to prevent excessive API calls.
🔹 **Concepts:** System Design, Concurrency, Rate Limiting.

```
import time

class RateLimiter:
    def __init__(self, max_requests, time_window):
        self.max_requests = max_requests
        self.time_window = time_window
        self.tokens = max_requests
        self.last_request_time = time.time()

    def allow_request(self):
        current_time = time.time()
        elapsed_time = current_time - self.last_request_time
        self.tokens = min(self.max_requests, self.tokens +
elapsed_time * (self.max_requests / self.time_window))
        self.last_request_time = current_time

        if self.tokens >= 1:
            self.tokens -= 1
            return True
        return False

# Example usage
limiter = RateLimiter(5, 10)  # 5 requests per 10 seconds
print(limiter.allow_request())  # Output: True
```

Flashcard 12: Predict House Prices Using Linear Regression

📌 **Problem:** Predict house prices using a simple Linear Regression model.
🔹 **Concepts:** Machine Learning, Linear Regression, scikit-learn.

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Example dataset (square footage vs. price)
X = np.array([[1400], [1600], [1700], [1875], [1100]])  # Feature:
Square footage
y = np.array([245000, 312000, 279000, 308000, 199000])  # Target:
House prices

# Train model
model = LinearRegression()
model.fit(X, y)

# Predict price for a 1500 sq ft house
predicted_price = model.predict([[1500]])
print(predicted_price)  # Output: Approximate house price
```

Flashcard 13: Generate Fibonacci Sequence (Recursion & Memoization)

📌 **Problem:** Generate Fibonacci numbers efficiently.
🔹 **Concepts:** Recursion, Memoization.

```
from functools import lru_cache

@lru_cache(maxsize=None)
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(10))  # Output: 55
```

Flashcard 14: Anomaly Detection Using Isolation Forest

📌 **Problem:** Detect anomalies in a dataset.
🔹 **Concepts:** Outlier detection, Machine Learning.

```
import numpy as np
from sklearn.ensemble import IsolationForest

X = np.array([[10], [12], [14], [1000], [16], [18]])  # 1000 is an
anomaly

clf = IsolationForest(contamination=0.2)
clf.fit(X)

anomalies = clf.predict(X)
print(anomalies)  # Expected output: [-1, 1, 1, -1, 1, 1] (where -1
indicates anomalies)
```

Flashcard 15: Deploy AI Model with FastAPI

📌 **Problem:** Deploy a simple AI model using FastAPI.
🔹 **Concepts:** Model Deployment, FastAPI.

```
from fastapi import FastAPI
import pickle
import numpy as np

app = FastAPI()

# Load pre-trained model
model = pickle.load(open("model.pkl", "rb"))

@app.post("/predict/")
def predict(features: list):
    prediction = model.predict([np.array(features)])
    return {"prediction": prediction.tolist()}

# Run server with: uvicorn script_name:app --reload
```

As an **AI Software Engineer**, you need to master a combination of **AI/ML techniques, software engineering principles, and system design**. Below are some essential **methods and concepts** you should know:

1️⃣ Core Machine Learning & AI Methods

✅ Supervised Learning

- Linear Regression, Logistic Regression
- Decision Trees, Random Forest
- Support Vector Machines (SVM)
- Gradient Boosting (XGBoost, LightGBM, CatBoost)

✅ Unsupervised Learning

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN (Density-Based Clustering)
- Principal Component Analysis (PCA), t-SNE

✅ Deep Learning

- Neural Networks (MLP, CNN, RNN, LSTMs, Transformers)
- Generative Models (GANs, VAEs)
- Transfer Learning (ResNet, BERT, GPT)

✅ Reinforcement Learning

- Q-Learning, Deep Q-Networks (DQN)
- Policy Gradient Methods (PPO, A3C)

2️⃣ Essential AI Engineering Methods

✅ Model Optimization & Deployment

- Quantization & Pruning (ONNX, TensorRT, TFLite)
- Model Distillation

- Latency Reduction Techniques

✅ MLOps & CI/CD for AI

- Model Tracking & Experimentation (MLflow, Weights & Biases)
- Automated Model Deployment (Docker, Kubernetes, FastAPI)
- Feature Store (Feast)

✅ Explainability & Fairness in AI

- SHAP (Shapley Values)
- LIME (Local Interpretable Model-agnostic Explanations)
- Bias Detection (Fairlearn, AIF360)

3️⃣ Advanced NLP & Computer Vision

✅ NLP Techniques

- Tokenization (BPE, WordPiece)
- Named Entity Recognition (NER)
- Text Summarization & Sentiment Analysis
- Large Language Models (LLMs) like GPT-4, LLaMA

✅ Computer Vision Techniques

- Object Detection (YOLO, Faster R-CNN)
- Image Segmentation (UNet, Mask R-CNN)
- OCR (Tesseract, EasyOCR)

4️⃣ System Design for AI

✅ Big Data Processing

- Apache Spark, Dask, Ray
- Kafka for real-time AI

✅ Scalable AI Model Serving

- TensorFlow Serving
- TorchServe
- Triton Inference Server

5️⃣ Key Software Engineering Skills

✅ Data Structures & Algorithms

- Graph Algorithms (Dijkstra, BFS, DFS)
- Dynamic Programming
- Hashing & Trie

✅ Design Patterns & Best Practices

- Microservices for AI
- Event-Driven Architecture

1️⃣ Built-in Python Methods

✅ String Methods

python
CopierModifier

```
s = "hello world"
s.upper()  # 'HELLO WORLD'
s.lower()  # 'hello world'
s.strip()  # Removes leading/trailing whitespace
s.replace("hello", "hi")  # 'hi world'
s.split(" ")  # ['hello', 'world']
```

✅ List Methods

python
CopierModifier

```
lst = [1, 2, 3]
lst.append(4)  # [1, 2, 3, 4]
lst.pop()  # Removes last element, returns 4
lst.remove(2)  # Removes the first occurrence of 2
lst.sort(reverse=True)  # Sorts in descending order
lst.index(3)  # Returns index of 3
```

✅ Dictionary Methods

python
CopierModifier

```
d = {"a": 1, "b": 2}
d.keys()  # dict_keys(['a', 'b'])
d.values()  # dict_values([1, 2])
d.items()  # dict_items([('a', 1), ('b', 2)])
d.get("a", 0)  # 1 (returns 0 if key doesn't exist)
d.update({"c": 3})  # {'a': 1, 'b': 2, 'c': 3}
```

✅ Set Methods

python
CopierModifier

```
s = {1, 2, 3}
s.add(4)  # {1, 2, 3, 4}
s.remove(2)  # Removes 2
s.union({3, 4, 5})  # {1, 2, 3, 4, 5}
s.intersection({2, 3})  # {3}
```

2️⃣ NumPy Methods (For Fast Computations)

```
python
CopierModifier
import numpy as np

arr = np.array([1, 2, 3, 4])
arr.mean()  # Mean value
arr.std()  # Standard deviation
np.dot(arr, arr)  # Dot product
np.linalg.inv(np.array([[1, 2], [3, 4]]))  # Matrix inverse
```

3️⃣ Pandas Methods (For Data Handling)

```
python
CopierModifier
import pandas as pd

df = pd.DataFrame({"A": [1, 2], "B": [3, 4]})
df.describe()  # Summary statistics
df.head()  # First 5 rows
df["A"].mean()  # Mean of column A
df.sort_values("B", ascending=False)  # Sort by column B
```

4️⃣ Scikit-Learn Methods (For Machine Learning)

```
python
CopierModifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
model = LogisticRegression()
model.fit(X_train, y_train)  # Train model
preds = model.predict(X_test)  # Make predictions
```

5️⃣ TensorFlow & PyTorch Methods (For Deep Learning)

```
python
CopierModifier
import tensorflow as tf
model = tf.keras.Sequential([tf.keras.layers.Dense(10,
activation='relu')])
model.compile(optimizer="adam", loss="mse")
model.fit(X_train, y_train, epochs=10)

import torch
tensor = torch.tensor([1.0, 2.0, 3.0])
tensor.mean()  # Compute mean
```

6️⃣ Random & OS Methods (For Utility)

```
python
CopierModifier
import random
random.randint(1, 10)  # Random integer between 1 and 10
random.choice(["apple", "banana", "cherry"])  # Random choice

import os
os.getcwd()  # Get current directory
os.listdir()  # List files in directory
```

Python Code Optimization Cheatsheet 🚀

Optimizing your code can make it **faster, more memory-efficient, and scalable**, especially for AI and ML tasks. Here's a **cheatsheet** covering key techniques!

1️⃣ General Python Performance Tips

✅ Use List Comprehensions Instead of Loops

🔴 Slow:

python
CopierModifier

```
squared = []
for i in range(10):
    squared.append(i ** 2)
```

🟢 Fast:

python
CopierModifier

```
squared = [i ** 2 for i in range(10)]
```

✅ Use Generators for Memory Efficiency

🔴 List (Consumes more memory):

python
CopierModifier

```
nums = [i for i in range(1000000)]
```

🟢 Generator (Lazily evaluates elements, saves memory):

python
CopierModifier

```
nums = (i for i in range(1000000))
```

✅ Use map() & filter() Instead of Loops

🔴 Slow:

```
python
CopierModifier
doubled = []
for i in range(10):
    doubled.append(i * 2)
```

🟢 Fast:

```
python
CopierModifier
doubled = list(map(lambda x: x * 2, range(10)))
```

🔴 Slow:

```
python
CopierModifier
evens = []
for i in range(10):
    if i % 2 == 0:
        evens.append(i)
```

🟢 Fast:

```
python
CopierModifier
evens = list(filter(lambda x: x % 2 == 0, range(10)))
```

2️⃣ Optimizing Data Structures

✅ **Use set Instead of list for Fast Lookups**
🔴 Slow (O(n) lookup):

```
python
CopierModifier
nums = [1, 2, 3, 4, 5]
print(3 in nums)  # Takes O(n)
```

🟢 Fast (O(1) lookup):

```
python
CopierModifier
nums = {1, 2, 3, 4, 5}
print(3 in nums)  # Takes O(1)
```

✅ **Use defaultdict for Efficient Dictionary Operations**
🔴 Slow:

```
python
CopierModifier
word_count = {}
for word in ["apple", "banana", "apple"]:
    if word in word_count:
        word_count[word] += 1
    else:
        word_count[word] = 1
```

🟢 Fast:

```
python
CopierModifier
from collections import defaultdict
word_count = defaultdict(int)
for word in ["apple", "banana", "apple"]:
    word_count[word] += 1
```

✅ **Use deque Instead of List for Fast Insertions/Deletions**
🔴 Slow (list.pop(0) takes O(n)):

```
python
CopierModifier
queue = [1, 2, 3, 4]
queue.pop(0)
```

🟢 Fast (deque.popleft() takes O(1)):

```
python
CopierModifier
from collections import deque
queue = deque([1, 2, 3, 4])
```

```
queue.popleft()
```

3️⃣ Optimizing Loops & Iterations

✅ Use zip() for Iterating Multiple Lists Efficiently

🔴 Slow:

```
python
CopierModifier
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
for i in range(len(names)):
    print(names[i], ages[i])
```

🟢 Fast:

```
python
CopierModifier
for name, age in zip(names, ages):
    print(name, age)
```

✅ Avoid Unnecessary Computation in Loops

🔴 Slow:

```
python
CopierModifier
for i in range(1000000):
    squared = i ** 2  # Recalculating every time
```

🟢 Fast:

```
python
CopierModifier
squared = [i ** 2 for i in range(1000000)]
```

4️⃣ Memory Optimization Tips

✅ Use sys.getsizeof() to Check Memory Usage

python
CopierModifier

```
import sys
lst = [i for i in range(1000)]
print(sys.getsizeof(lst))  # Memory usage in bytes
```

✅ Use slots in Classes to Reduce Memory Overhead

🔴 Normal Class (Takes More Memory):

python
CopierModifier

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

🟢 Optimized with __slots__ (Takes Less Memory):

python
CopierModifier

```
class Person:
    __slots__ = ['name', 'age']
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

✅ Use del to Free Memory

python
CopierModifier

```
import gc
data = [i for i in range(1000000)]
del data  # Removes reference
gc.collect()  # Forces garbage collection
```

5️⃣ Speeding Up Computations

✅ **Use NumPy for Faster Array Operations**
🔴 Slow (Using Lists):

python
CopierModifier
```
lst = [i * 2 for i in range(1000000)]
```

🟢 Fast (Using NumPy Arrays):

python
CopierModifier
```
import numpy as np
arr = np.arange(1000000) * 2
```

✅ **Use multiprocessing for Parallel Processing**

python
CopierModifier
```
from multiprocessing import Pool

def square(n):
    return n * n

with Pool(4) as p:  # Use 4 CPU cores
    results = p.map(square, range(10))
```

✅ **Use cython or numba for Faster Computation**
🔴 Slow Python Loop:

python
CopierModifier
```
def slow_function(n):
    result = 0
    for i in range(n):
        result += i
    return result
```

🟢 Fast with numba:

python

```
CopierModifier
from numba import jit

@jit(nopython=True)
def fast_function(n):
    result = 0
    for i in range(n):
        result += i
    return result
```

6️⃣ Optimize AI & ML Code

✅ Vectorize Operations Instead of Using Loops 🔴 Slow:

```
python
CopierModifier
import numpy as np
X = np.random.rand(1000000)
Y = np.zeros(1000000)
for i in range(len(X)):
    Y[i] = X[i] * 2 + 3
```

🟢 Fast:

```
python
CopierModifier
Y = X * 2 + 3
```

✅ Use joblib for Parallel ML Processing

```
python
CopierModifier
from joblib import Parallel, delayed
import numpy as np

def process(i):
    return i ** 2

results = Parallel(n_jobs=4)(delayed(process)(i) for i in range(10))
```

🔥 Final Cheat Sheet Summary

Optimization	Tip
List Comprehension	Use [x for x in iterable] instead of loops
Set for Lookup	Use {} instead of list for faster membership tests
Use Generators	(x for x in iterable) saves memory
Use NumPy & Pandas	Faster matrix operations & data handling
Use Multiprocessing	multiprocessing.Pool for parallel tasks
Use Numba & Cython	JIT compilation for faster execution
Use defaultdict & deque	More efficient dictionary & queue operations