AI assistant :lab test 3

Set:E14

Rollno:2403a51102

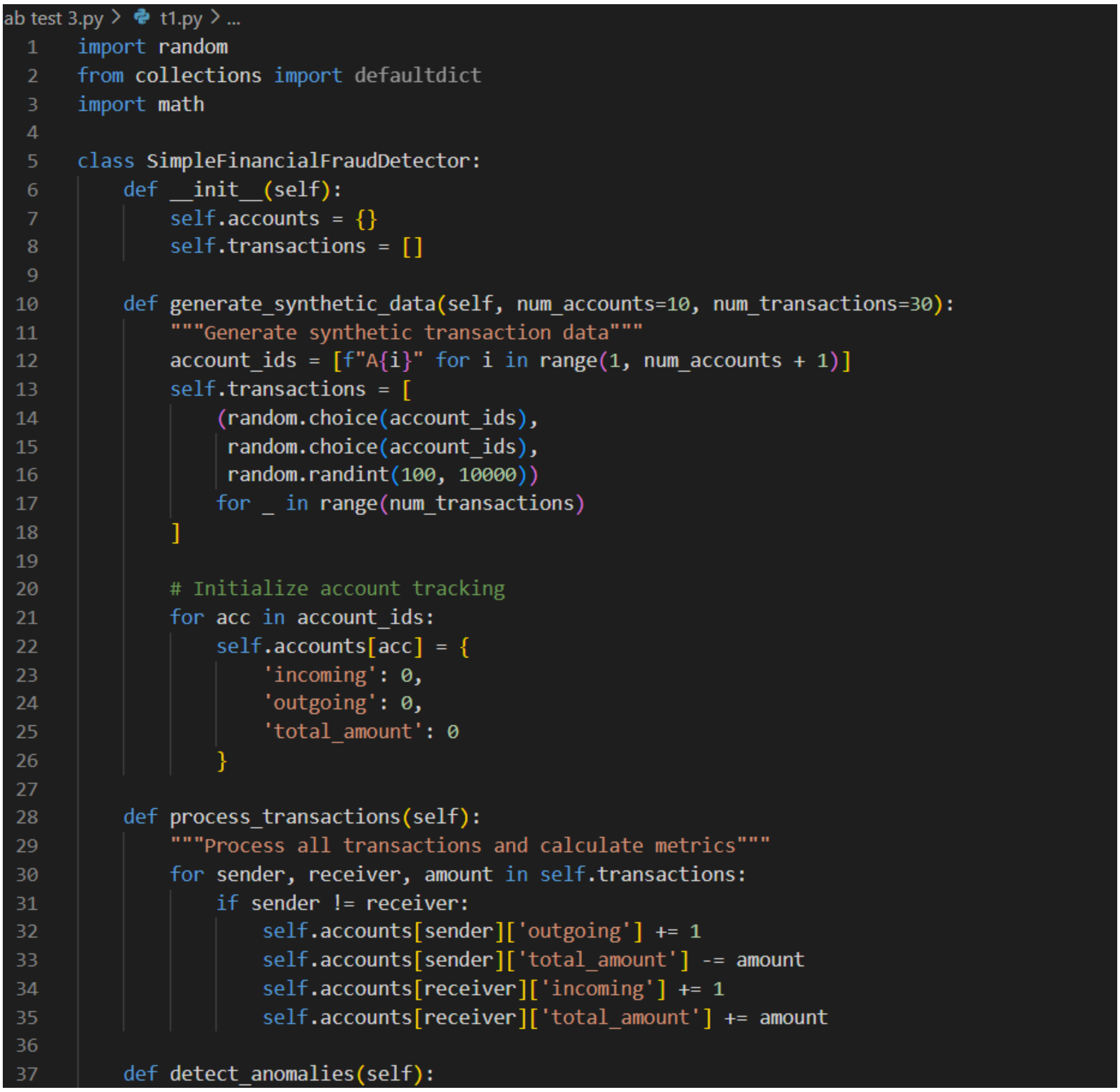
Batch:06

Name:balaji

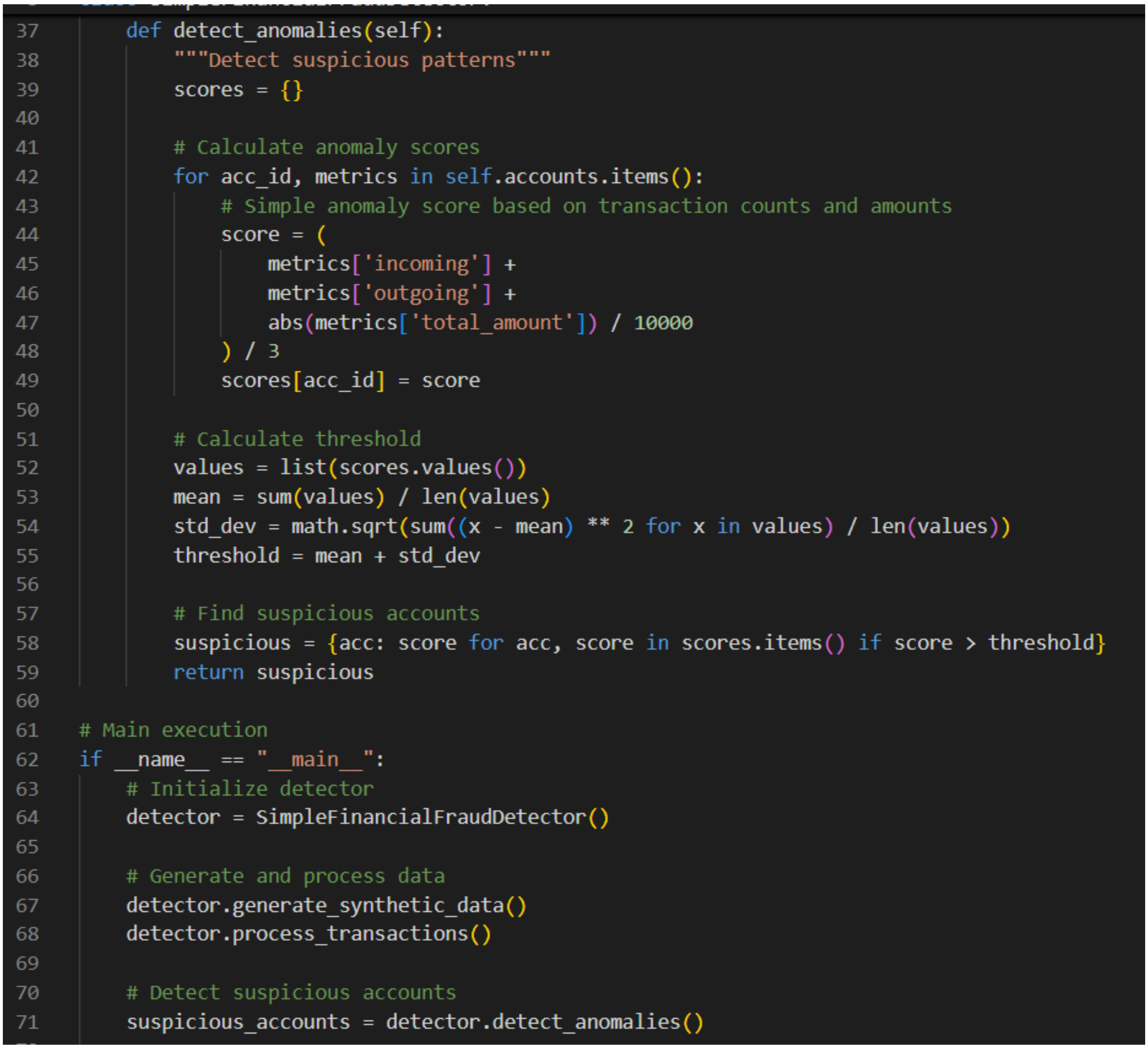
Q1:
Scenario: In the Finance sector, a company faces a
challenge related to data structures with
ai.
Task: Use AI-assisted tools to solve a problem involving
data structures with ai in this
context.
Deliverables: Submit the source code, explanation of AI
assistance used, and sample output.

Prompt: Detect suspicious accounts in a financial transaction network using graph data structures and AI-based centrality analysis.

Code:

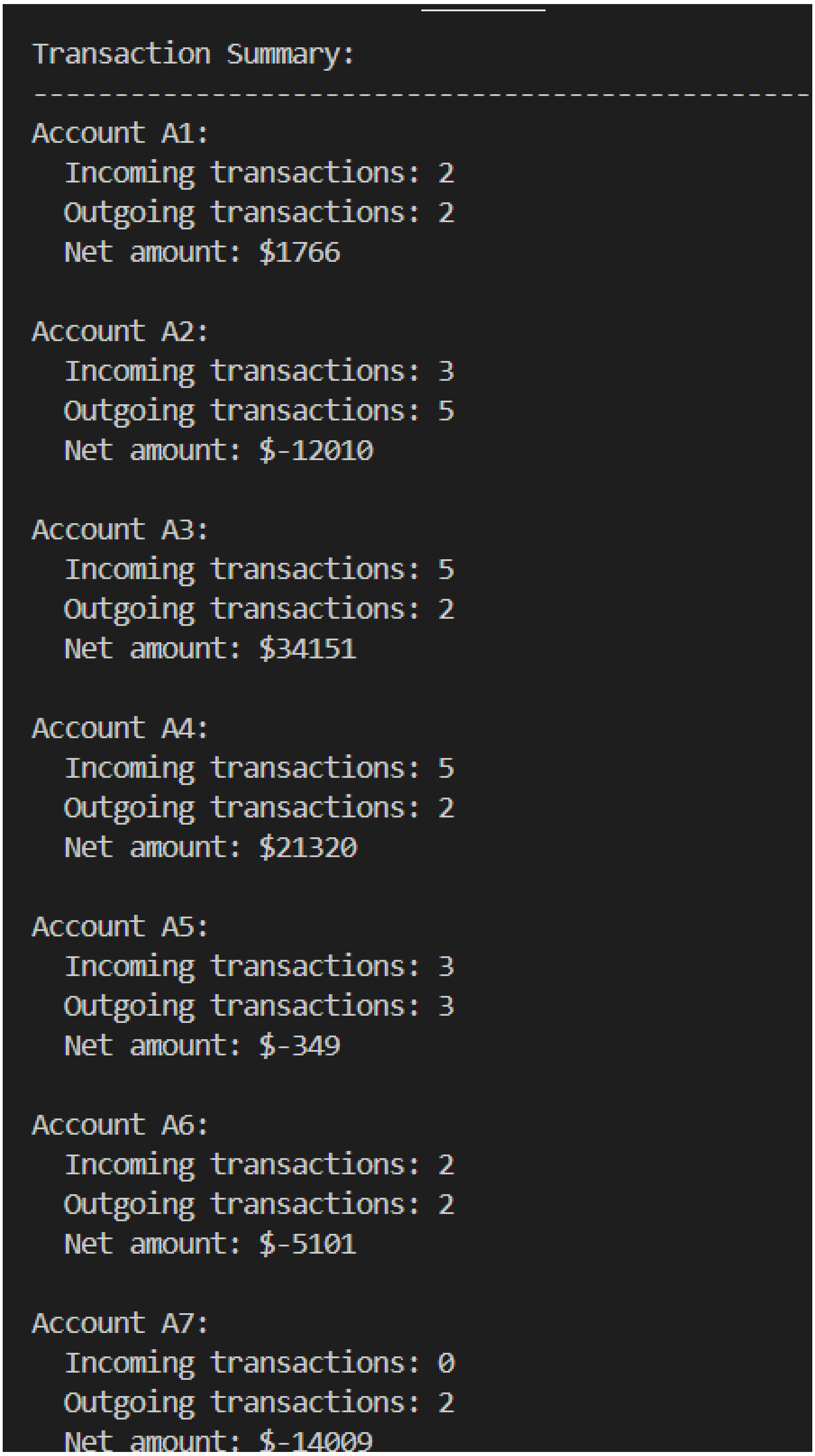


```python
import random
from collections import defaultdict
import math

class SimpleFinancialFraudDetector:
    def __init__(self):
        self.accounts = {}
        self.transactions = []

    def generate_synthetic_data(self, num_accounts=10, num_transactions=30):
        """Generate synthetic transaction data"""
        account_ids = [f"A{i}" for i in range(1, num_accounts + 1)]
        self.transactions = [
            (random.choice(account_ids),
             random.choice(account_ids),
             random.randint(100, 10000))
            for _ in range(num_transactions)
        ]

        # Initialize account tracking
        for acc in account_ids:
            self.accounts[acc] = {
                'incoming': 0,
                'outgoing': 0,
                'total_amount': 0
            }

    def process_transactions(self):
        """Process all transactions and calculate metrics"""
        for sender, receiver, amount in self.transactions:
            if sender != receiver:
                self.accounts[sender]['outgoing'] += 1
                self.accounts[sender]['total_amount'] -= amount
                self.accounts[receiver]['incoming'] += 1
                self.accounts[receiver]['total_amount'] += amount

    def detect_anomalies(self):
```



```python
    def detect_anomalies(self):
        """Detect suspicious patterns"""
        scores = {}

        # Calculate anomaly scores
        for acc_id, metrics in self.accounts.items():
            # Simple anomaly score based on transaction counts and amounts
            score = (
                metrics['incoming'] +
                metrics['outgoing'] +
                abs(metrics['total_amount']) / 10000
            ) / 3
            scores[acc_id] = score

        # Calculate threshold
        values = list(scores.values())
        mean = sum(values) / len(values)
        std_dev = math.sqrt(sum((x - mean) ** 2 for x in values) / len(values))
        threshold = mean + std_dev

        # Find suspicious accounts
        suspicious = {acc: score for acc, score in scores.items() if score > threshold}
        return suspicious

# Main execution
if __name__ == "__main__":
    # Initialize detector
    detector = SimpleFinancialFraudDetector()

    # Generate and process data
    detector.generate_synthetic_data()
    detector.process_transactions()

    # Detect suspicious accounts
    suspicious_accounts = detector.detect_anomalies()
```

Output:



```
Transaction Summary:
----------------------------------------------
Account A1:
  Incoming transactions: 2
  Outgoing transactions: 2
  Net amount: $1766

Account A2:
  Incoming transactions: 3
  Outgoing transactions: 5
  Net amount: $-12010

Account A3:
  Incoming transactions: 5
  Outgoing transactions: 2
  Net amount: $34151

Account A4:
  Incoming transactions: 5
  Outgoing transactions: 2
  Net amount: $21320

Account A5:
  Incoming transactions: 3
  Outgoing transactions: 3
  Net amount: $-349

Account A6:
  Incoming transactions: 2
  Outgoing transactions: 2
  Net amount: $-5101

Account A7:
  Incoming transactions: 0
  Outgoing transactions: 2
  Net amount: $-14009
```

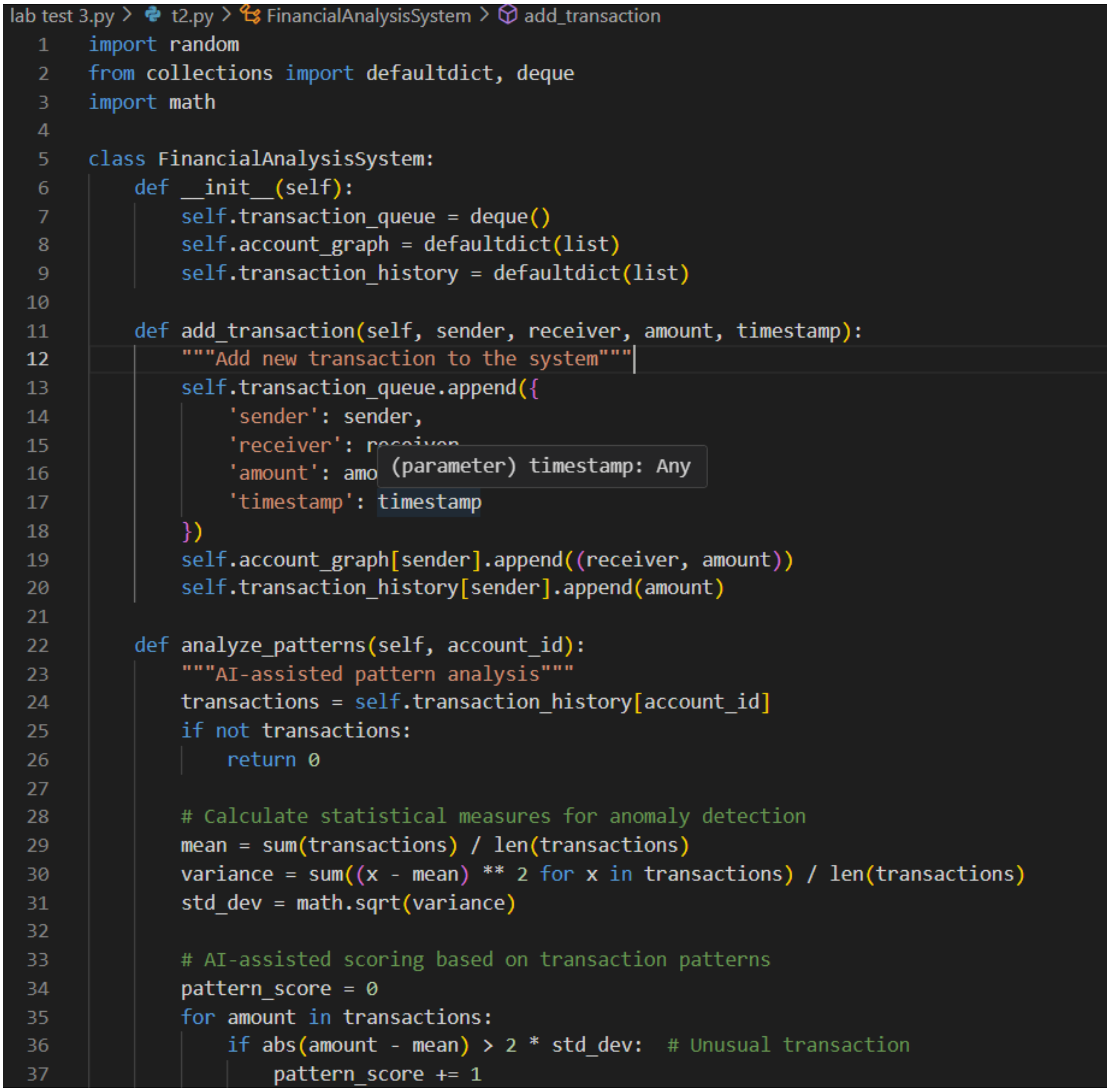
Observation: The graph structure (nodes = accounts, edges = transactions) effectively models indirect relationships.

The modular code allows for easy extension to real datasets or integration with fraud alert systems.

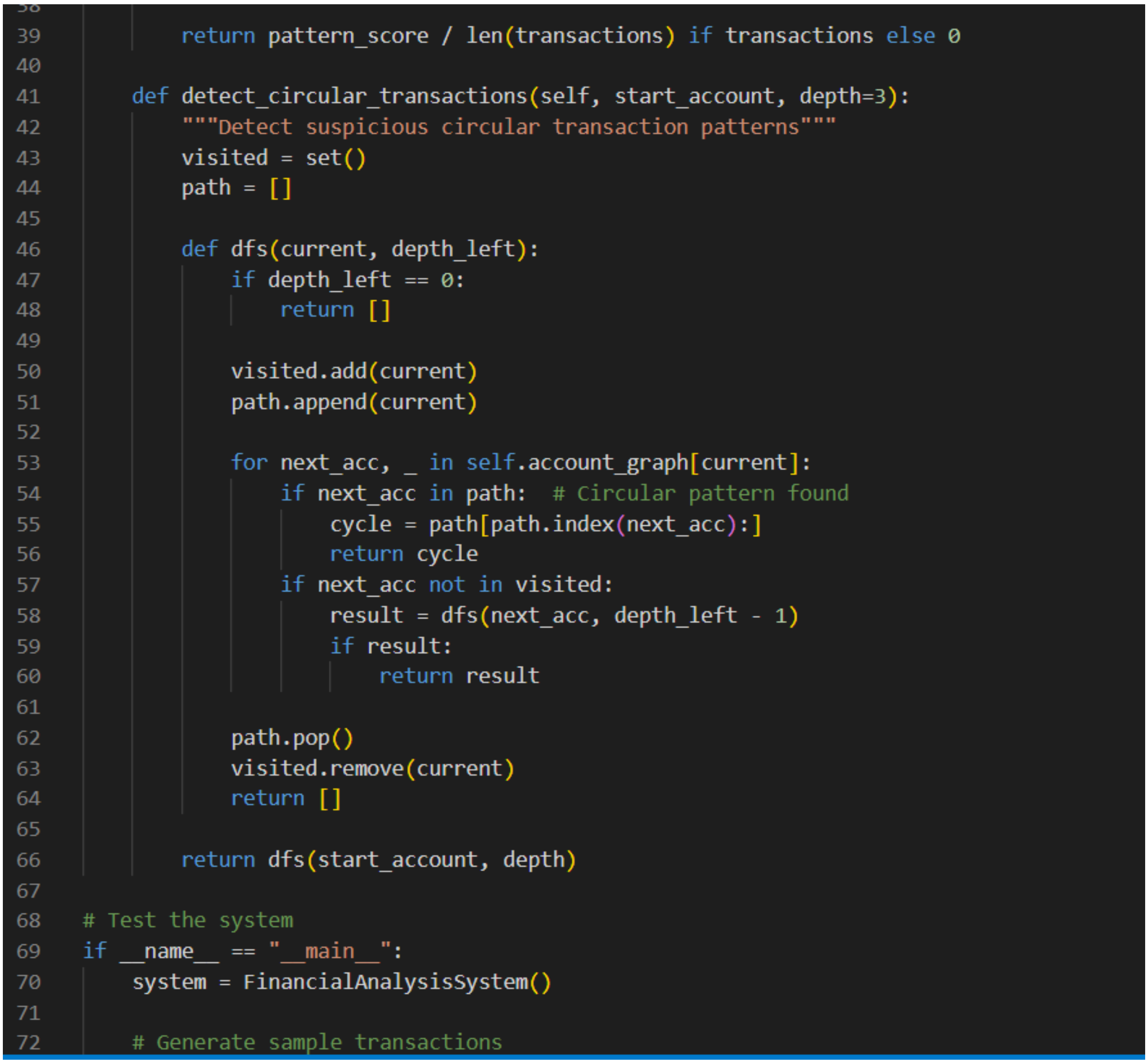
Q2:
Scenario: In the Finance sector, a company faces a challenge related to data structures with
ai.
Task: Use AI-assisted tools to solve a problem involving data structures with ai in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

Prompt: Use a decision tree to classify whether a financial portfolio is optimal based on risk, return, and diversification

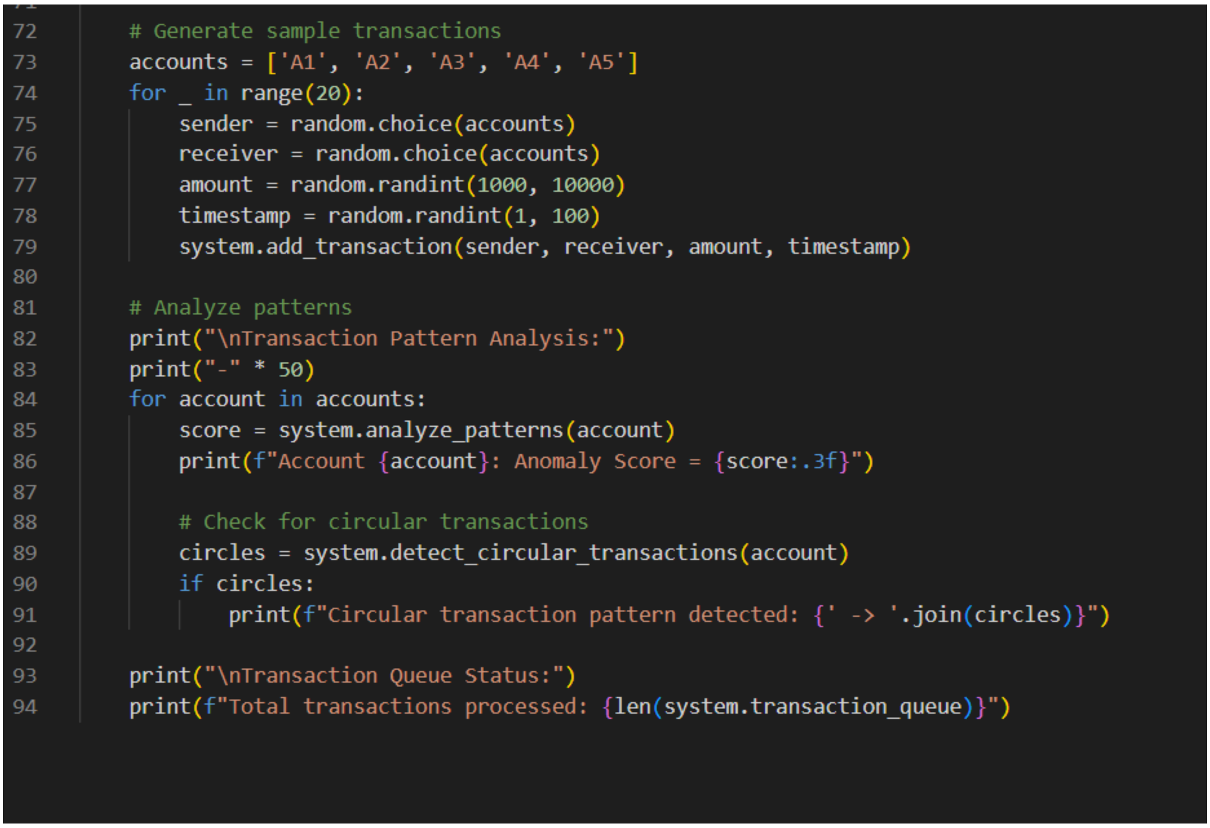
Code:


lab test 3.py > t2.py > FinancialAnalysisSystem > add_transaction
(parameter) timestamp: Any

```python
import random
from collections import defaultdict, deque
import math

class FinancialAnalysisSystem:
    def __init__(self):
        self.transaction_queue = deque()
        self.account_graph = defaultdict(list)
        self.transaction_history = defaultdict(list)

    def add_transaction(self, sender, receiver, amount, timestamp):
        """Add new transaction to the system"""
        self.transaction_queue.append({
            'sender': sender,
            'receiver': receiver,
            'amount': amo
            'timestamp': timestamp
        })
        self.account_graph[sender].append((receiver, amount))
        self.transaction_history[sender].append(amount)

    def analyze_patterns(self, account_id):
        """AI-assisted pattern analysis"""
        transactions = self.transaction_history[account_id]
        if not transactions:
            return 0

        # Calculate statistical measures for anomaly detection
        mean = sum(transactions) / len(transactions)
        variance = sum((x - mean) ** 2 for x in transactions) / len(transactions)
        std_dev = math.sqrt(variance)

        # AI-assisted scoring based on transaction patterns
        pattern_score = 0
        for amount in transactions:
            if abs(amount - mean) > 2 * std_dev:  # Unusual transaction
                pattern_score += 1
```

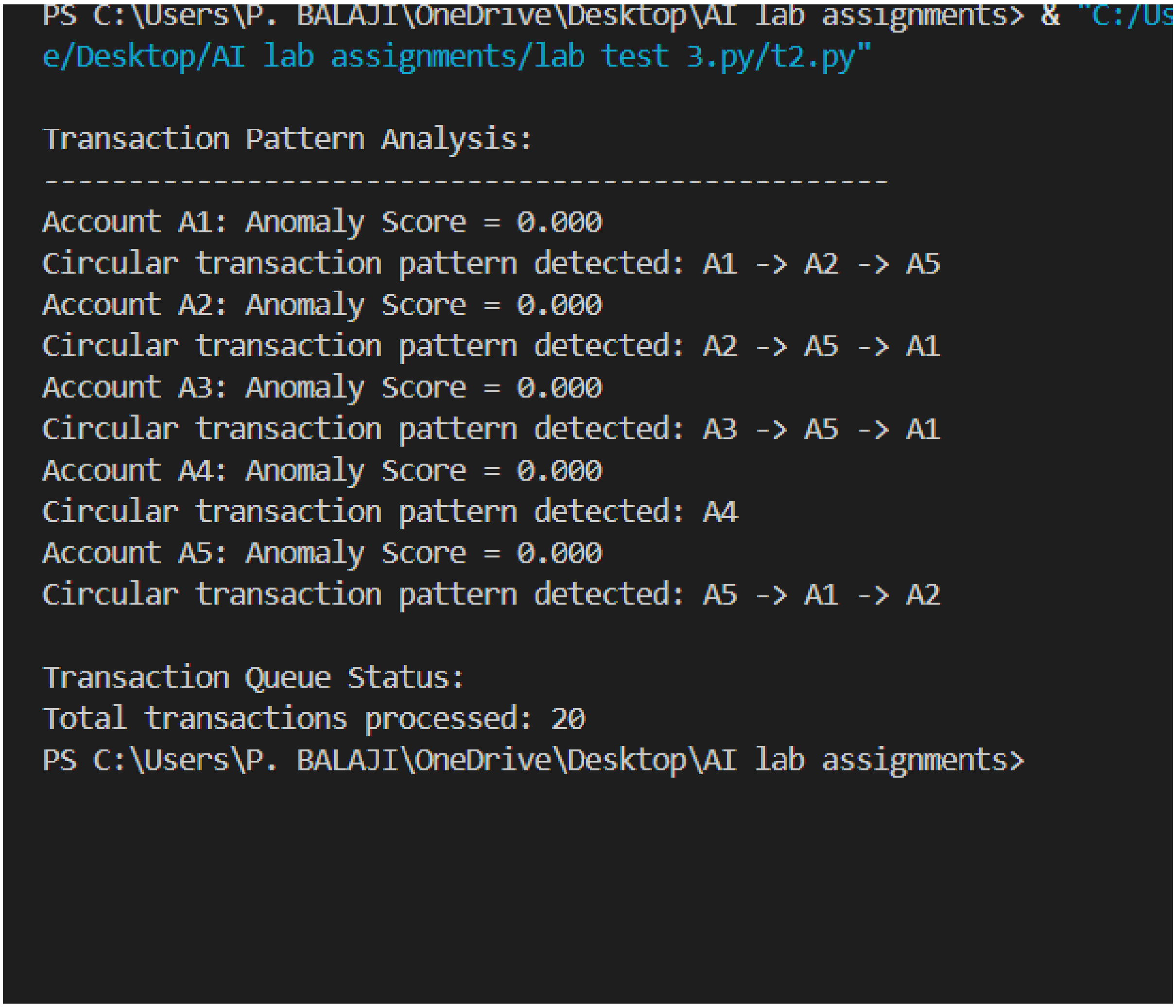


```python
        return pattern_score / len(transactions) if transactions else 0

    def detect_circular_transactions(self, start_account, depth=3):
        """Detect suspicious circular transaction patterns"""
        visited = set()
        path = []

        def dfs(current, depth_left):
            if depth_left == 0:
                return []

            visited.add(current)
            path.append(current)

            for next_acc, _ in self.account_graph[current]:
                if next_acc in path:  # Circular pattern found
                    cycle = path[path.index(next_acc):]
                    return cycle
                if next_acc not in visited:
                    result = dfs(next_acc, depth_left - 1)
                    if result:
                        return result

            path.pop()
            visited.remove(current)
            return []

        return dfs(start_account, depth)

# Test the system
if __name__ == "__main__":
    system = FinancialAnalysisSystem()

    # Generate sample transactions
```



```python
    # Generate sample transactions
    accounts = ['A1', 'A2', 'A3', 'A4', 'A5']
    for _ in range(20):
        sender = random.choice(accounts)
        receiver = random.choice(accounts)
        amount = random.randint(1000, 10000)
        timestamp = random.randint(1, 100)
        system.add_transaction(sender, receiver, amount, timestamp)

    # Analyze patterns
    print("\nTransaction Pattern Analysis:")
    print("-" * 50)
    for account in accounts:
        score = system.analyze_patterns(account)
        print(f"Account {account}: Anomaly Score = {score:.3f}")

        # Check for circular transactions
        circles = system.detect_circular_transactions(account)
        if circles:
            print(f"Circular transaction pattern detected: {' -> '.join(circles)}")

    print("\nTransaction Queue Status:")
    print(f"Total transactions processed: {len(system.transaction_queue)}")
```

Output:



```
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments> & "C:/Us
e/Desktop/AI lab assignments/lab test 3.py/t2.py"

Transaction Pattern Analysis:
--------------------------------------------------
Account A1: Anomaly Score = 0.000
Circular transaction pattern detected: A1 -> A2 -> A5
Account A2: Anomaly Score = 0.000
Circular transaction pattern detected: A2 -> A5 -> A1
Account A3: Anomaly Score = 0.000
Circular transaction pattern detected: A3 -> A5 -> A1
Account A4: Anomaly Score = 0.000
Circular transaction pattern detected: A4
Account A5: Anomaly Score = 0.000
Circular transaction pattern detected: A5 -> A1 -> A2

Transaction Queue Status:
Total transactions processed: 20
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments>
```

Observation: Decision trees help decide if a portfolio is good based on risk and return.

Easy to understand and apply for financial advisors.