Question 1: Mandatory : Elaborate what your internship or academic projects were?

a. **What did the system do?**
b. **What other systems have you seen in the wild like that?**
c. **How do you approach the development problem?**
d. **What were interesting aspects where you copied code from Stack Overflow?**
e. **What did you learn from some very specific copy paste? Mention explicitly some of them.**

Answers:

a. "Ease My Order" is a Restaurant Order & Billing System. It allows users to browse the menu, place orders, and automatically generated bills.

b. Similar systems include restaurant POS (Point of Sale) systems. These systems also manage orders, billing, and sometimes inventory management.

c. I started by defining the system's requirements and functionalities. Then, I used Python and data processing libraries like Pandas, NumPy, and Matplotlib to handle data integration and analysis from CSV files. I tested each feature thoroughly to ensure accuracy and efficiency.

d. I found useful snippets for reading and processing CSV files efficiently and handling user inputs for order placement. I used code for implementing a simple menu driven interface and data visualization techniques for generating bill summaries.

e. From copying code for reading CSV files, I learned about handling various data formats and ensuring data integrity. Copying snippets for menu interfaces taught me about structuring user interactions clearly. These experiences helped me understand the importance of optimizing code for better performance and user experience.

Question 3: Write a function to parse any valid json string into a corresponding Object, List, or Map object. You can use C,C++, Java, Scala, Kotlin, Python, Node. Note that the integer and floating point should be arbitrary precision.

Answer: The Python function `parse_json` takes a string formatted in JSON and tries to convert it into a Python object such as a dictionary, list, or other basic data types. It uses Python's built-in `json` module to do this. The function is set up to handle numbers with very high precision, with `Decimal` class from the `decimal` module. If the JSON string is formatted correctly, it returns the converted Python object. If there's an error in the JSON formatting, it catches this and returns a message saying the JSON is invalid, showing what went wrong. This makes it easier to debug issues with the JSON data we're working with.

Code:

```
import json
from decimal import Decimal

def parse_json(json_string):
    try:
        result = json.loads(json_string, parse_float=Decimal,
parse_int=Decimal)
        return result
    except json.JSONDecodeError as e:
        return f"Invalid JSON: {str(e)}"

json_string = '{"name": "John", "age": 30, "balance": 12345.67}'
print("Valid JSON example:")
result = parse_json(json_string)
print(result)

invalid_json_string = '{"name": "John", "age": 30, "balance": 12345.67'
print("\nInvalid JSON example:")
result = parse_json(invalid_json_string)
print(result)
```

Question 5: Banking works by transferring money from account A to account B. Most of the time account A is in one bank while account B is another bank. Implement the code to transfer money. Remember, payee's code runs on a different computer than that of the receiver's code.

1. **What are the issues in such a system?**
2. **What can we do to mitigate some of the issues ?**
3. **Write the fixing yourself to demonstrate the mitigations.**

Answer: The code is about transferring money between two bank accounts. One account is at one bank, and the other is at a different bank. The code handles deposits and withdrawals and ensures transactions are secure even when processed on different computers.

1: Issues:-

a. Concurrency: When two transactions happen simultaneously, they might interfere with each other.
b. Rollback: If a deposit fails, we need to ensure the withdrawn amount is returned.
c. Security: Ensuring the transaction is secure from fraud or errors.

2: Mitigations:-

a. Threading: Using threads to handle multiple transactions simultaneously but carefully managing their interactions.
b. Atomic Transactions: Ensuring that if any part of the transaction fails, the entire transaction is rolled back.
c. Logging: Keeping a log of transactions for troubleshooting and security checks.

3: Fixing:- Added locks to prevent concurrency issues and ensured transactions roll back if any part fails.

Code:

```
import threading
import time

class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance
        self.lock = threading.Lock()

    def deposit(self, amount):
        with self.lock:
            self.balance += amount
            return True

    def withdraw(self, amount):
        with self.lock:
            if self.balance >= amount:
                self.balance -= amount
                return True
            return False

def transfer_money(from_account, to_account, amount):
    transaction_id = time.time()
    print(f"Trying to transfer ${amount} from {from_account.owner} to
{to_account.owner} with transaction ID: {transaction_id}")

    if from_account.withdraw(amount):
        if to_account.deposit(amount):
            print(f"Success: ${amount} was transferred from
{from_account.owner} to {to_account.owner}")
        else:
            print(f"Failed to deposit in {to_account.owner}'s account. Rolling
back transaction.")
            from_account.deposit(amount)
    else:
        print(f"Failed: Not enough money in {from_account.owner}'s account to
transfer.")

account_A = BankAccount('User A', 1000)
account_B = BankAccount('User B', 500)

transfer1 = threading.Thread(target=transfer_money, args=(account_A,
account_B, 200))
transfer2 = threading.Thread(target=transfer_money, args=(account_B,
account_A, 150))
```

```
transfer1.start()
transfer2.start()

transfer1.join()
transfer2.join()
```