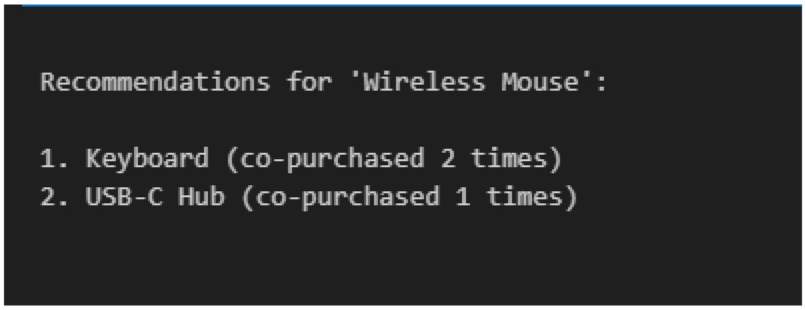
Q1:

```
transactionsLog = [
    {'orderId': 1001, 'customerId': 'cust_Ahmed', 'productId':
'prod_10'},
    {'orderId': 1001, 'customerId': 'cust_Ahmed', 'productId':
'prod_12'},
    {'orderId': 1002, 'customerId': 'cust_Bisma', 'productId':
'prod_15'},
    {'orderId': 1003, 'customerId': 'cust_Ahmed', 'productId':
'prod_15'},
    {'orderId': 1004, 'customerId': 'cust_Faisal', 'productId':
'prod_12'},
    {'orderId': 1004, 'customerId': 'cust_Faisal', 'productId':
'prod_10'},
]


productCatalog = {
    'prod_10': 'Wireless Mouse',
    'prod_12': 'Keyboard',
    'prod_15': 'USB-C Hub',
}


customerData = {}


def processTransactions(transactionList):
    for t in transactionList:
        customer = t['customerId']
        product = t['productId']
        if customer not in customerData:
            customerData[customer] = set()
        customerData[customer].add(product)
    return customerData

def findFrequentPairs(customerData):
    CountPairs = {}
    for products in customerData.values():
        productList = list(products)
        for i in range(len(productList)):
            for j in range(i + 1, len(productList)):
                p1, p2 = sorted([productList[i], productList[j]])
                key = (p1, p2)
                CountPairs[key] = CountPairs.get(key, 0) + 1
    return CountPairs

def getRecommendations(targetProductId, pairs):
    recommendations = []
    for (p1, p2), count in pairs.items():
        if targetProductId in (p1, p2):
            other = p2 if p1 == targetProductId else p1
            recommendations.append((other, count))
    recommendations.sort(key=lambda x: x[1], reverse=True)
    return recommendations

def generateReport(targetProductId, recommendations, catalog):
    targetName = catalog.get(targetProductId, targetProductId)
    print(f"\nRecommendations for '{targetName}':\n")
    if not recommendations:
        print("No recommendations available.")
        return
    for idx, (prodId, count) in enumerate(recommendations, start=1):
        prodName = catalog.get(prodId, prodId)
        print(f"{idx}. {prodName} (co-purchased {count} times)")

customerData = processTransactions(transactionsLog)
pairs = findFrequentPairs(customerData)
recommendations = getRecommendations('prod_10', pairs)
generateReport('prod_10', recommendations, productCatalog)
```

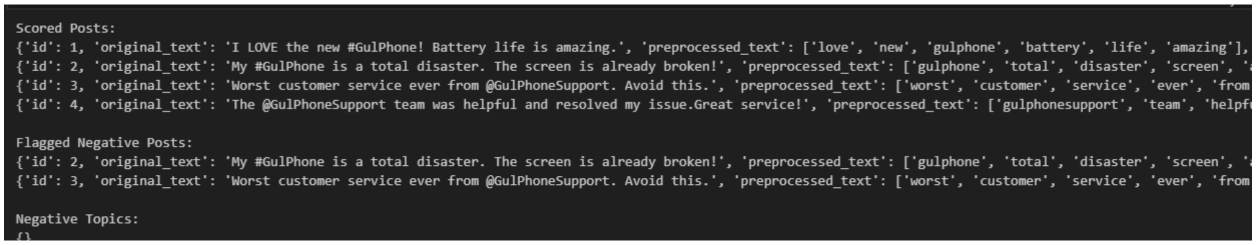


```
Recommendations for 'Wireless Mouse':

1. Keyboard (co-purchased 2 times)
2. USB-C Hub (co-purchased 1 times)
```

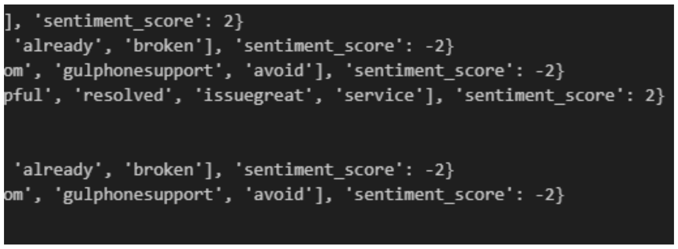
Q2:

```
allPosts = [{'id': 1,'text': 'I LOVE the new #GulPhone! Battery life is amazing.'},
            {'id': 2,'text': 'My #GulPhone is a total disaster. The screen is already broken!'},
            {'id': 3,'text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.'},
            {'id': 4,'text': 'The @GulPhoneSupport team was helpful and resolved my issue.Great service!'}]
```

```
PUNCTUATION_CHAR = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

STOPWORDS_SET =
{'i','me','my','a','an','the','is','am','was','and','but','if','or','t
o','of','at','by','for','with','this','that'}

POSITIVE_WORDS_SET = {'love','amazing','great','helpful','resolved'}

NEGATIVE_WORDS_SET = {'disaster','broken','worst','avoid','bad'}


def preprocessText(text, punctuationList, stopwordsSet):

    text = text.lower()

    for char in punctuationList:

        text = text.replace(char, "")

    words = text.split()

    return [w for w in words if w not in stopwordsSet]


def analyzePosts(postsList, punctuation, stopwords, positive,
negative):

    results = []

    preprocessed_texts = [preprocessText(post['text'], punctuation,
stopwords) for post in postsList]

    for post, words in zip(postsList, preprocessed_texts):

        score = sum(1 for w in words if w in positive) - sum(1 for w
in words if w in negative)

        results.append({

            'id': post['id'],

            'original_text': post['text'],

            'preprocessed_text': words,

            'sentiment_score': score
        })
    return results


def getFlaggedPosts(scoredPosts, sentimentThreshold=-1):
    return [p for p in scoredPosts if p['sentiment_score'] <=
sentimentThreshold]


def findNegativeTopics(flaggedPosts):
    topics = {}
    for post in flaggedPosts:
        for word in post['preprocessed_text']:
            if word.startswith('#') or word.startswith('@'):
                topics[word] = topics.get(word, 0) + 1
    return topics
scored = analyzePosts(allPosts, PUNCTUATION_CHAR, STOPWORDS_SET,
POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
flagged = getFlaggedPosts(scored, sentimentThreshold=-1)
topics = findNegativeTopics(flagged)
print("Scored Posts:")
for s in scored:
    print(s)


print("\nFlagged Negative Posts:")
for f in flagged:
    print(f)


print("\nNegative Topics:")
print(topics)
```



```
Scored Posts:
{'id': 1, 'original_text': 'I LOVE the new #GulPhone! Battery life is amazing.', 'preprocessed_text': ['love', 'new', 'gulphone', 'battery', 'life', 'amazing'],
{'id': 2, 'original_text': 'My #GulPhone is a total disaster. The screen is already broken!', 'preprocessed_text': ['gulphone', 'total', 'disaster', 'screen', '
{'id': 3, 'original_text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.', 'preprocessed_text': ['worst', 'customer', 'service', 'ever', 'from
{'id': 4, 'original_text': 'The @GulPhoneSupport team was helpful and resolved my issue.Great service!', 'preprocessed_text': ['gulphonesupport', 'team', 'helpf

Flagged Negative Posts:
{'id': 2, 'original_text': 'My #GulPhone is a total disaster. The screen is already broken!', 'preprocessed_text': ['gulphone', 'total', 'disaster', 'screen', '
{'id': 3, 'original_text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.', 'preprocessed_text': ['worst', 'customer', 'service', 'ever', 'from

Negative Topics:
{}
```

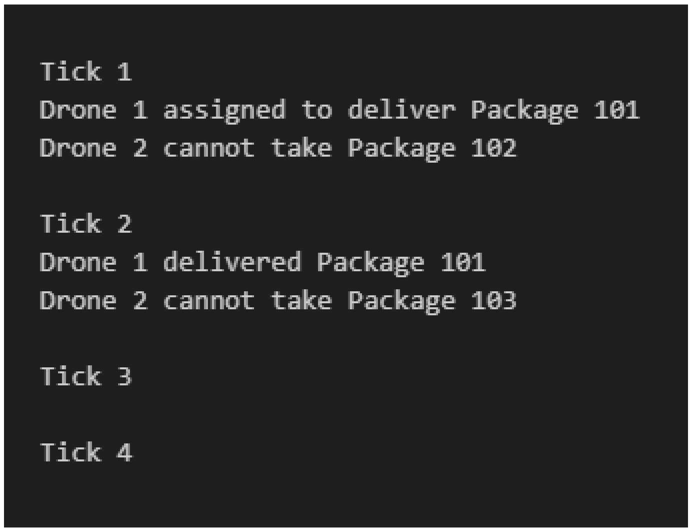


```
], 'sentiment_score': 2}
 'already', 'broken'], 'sentiment_score': -2}
om', 'gulphonesupport', 'avoid'], 'sentiment_score': -2}
pful', 'resolved', 'issuegreat', 'service'], 'sentiment_score': 2}


 'already', 'broken'], 'sentiment_score': -2}
om', 'gulphonesupport', 'avoid'], 'sentiment_score': -2}
```

Q3:

```
class Package:

    def __init__(self, packageId, weightInKg):

        self.packageId = packageId

        self.weightInKg = weightInKg

        self.isDelivered = False


    def markDelivered(self):

        self.isDelivered = True


class Drone:

    def __init__(self, droneId, maxLoadInKg):

        self.droneId = droneId

        self.maxLoadInKg = maxLoadInKg

        self.__status = 'idle'

        self.assignedPackage = None
    def getStatus(self):

        return self.__status


    def setStatus(self, newStatus):

        validStatus = ['idle', 'delivering', 'charging']

        if newStatus in validStatus:

            self.__status = newStatus

        else:

            print(f"Invalid status '{newStatus}' for Drone
{self.droneId}")


    def assignPackage(self, package):

        if self.__status == 'idle' and package.weightInKg <=
self.maxLoadInKg:

            self.assignedPackage = package

            self.setStatus('delivering')

            print(f"Drone {self.droneId} assigned to deliver Package
{package.packageId}")

        else:

            print(f"Drone {self.droneId} cannot take Package
{package.packageId}")


    def simulateTick(self):

        if self.__status == 'delivering':

            self.assignedPackage.markDelivered()

            print(f"Drone {self.droneId} delivered Package
{self.assignedPackage.packageId}")

            self.assignedPackage = None
            self.setStatus('charging')
        elif self.__status == 'charging':
            self.setStatus('idle')

class FleetManager:
    def __init__(self):
        self.drones = []
        self.pendingPackages = []

    def addDrone(self, drone):
        self.drones.append(drone)

    def addPackage(self, package):
        self.pendingPackages.append(package)

    def dispatchDrones(self):
        for drone in self.drones:
            if drone.getStatus() == 'idle' and self.pendingPackages:
                pkg = self.pendingPackages.pop(0)
                drone.assignPackage(pkg)

    def simulateTick(self):
        for drone in self.drones:
            drone.simulateTick()
        self.dispatchDrones()
```

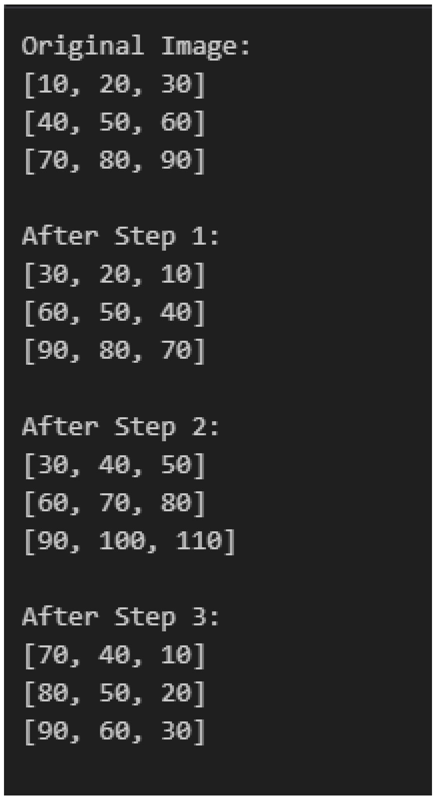
```
fm = FleetManager()


d1 = Drone(1, 10)

d2 = Drone(2, 5)

fm.addDrone(d1)

fm.addDrone(d2)


p1 = Package(101, 4)

p2 = Package(102, 8)

p3 = Package(103, 6)

fm.addPackage(p1)

fm.addPackage(p2)

fm.addPackage(p3)


for tick in range(4):

    print(f"\nTick {tick+1}")

    fm.simulateTick()
```



```
Tick 1
Drone 1 assigned to deliver Package 101
Drone 2 cannot take Package 102

Tick 2
Drone 1 delivered Package 101
Drone 2 cannot take Package 103

Tick 3

Tick 4
```

Q4:

```
class Image:

    def __init__(self, pixels):

        self.pixels = pixels


    def applyTransformation(self, func):

        new_pixels = func(self.pixels)

        return Image(new_pixels)


    def getCopy(self):

        return Image([row[:] for row in self.pixels])


    def __str__(self):

        return "\n".join(str(row) for row in self.pixels)


def flipHorizontal(data):

    return [row[::-1] for row in data]


def adjustBrightness(data, value):

    return [[max(0, min(255, p + value)) for p in row] for row in
data]


def rotate90(data):

    return [list(col) for col in zip(*data[::-1])]


class AugmentationPipeline:

    def __init__(self):

        self.steps = []
    def addStep(self, func):
        self.steps.append(func)

    def processImage(self, image):
        return [image.applyTransformation(f) for f in self.steps]

img = Image([
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
])

pipe = AugmentationPipeline()
pipe.addStep(flipHorizontal)
pipe.addStep(lambda d: adjustBrightness(d, 20))
pipe.addStep(rotate90)

results = pipe.processImage(img)

print("Original Image:")
print(img)
for i, res in enumerate(results, 1):
    print(f"\nAfter Step {i}:")
    print(res)
```



```
Original Image:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]

After Step 1:
[30, 20, 10]
[60, 50, 40]
[90, 80, 70]

After Step 2:
[30, 40, 50]
[60, 70, 80]
[90, 100, 110]

After Step 3:
[70, 40, 10]
[80, 50, 20]
[90, 60, 30]
```