**QUESTION NO 1:**

class Recipe:

def \_\_init\_\_(self, name, ingredients, instructions, dietary\_restrictions=None):

self.name = name

self.ingredients = ingredients

self.instructions = instructions

self.dietary\_restrictions = dietary\_restrictions if dietary\_restrictions else []

def add\_ingredient(self, ingredient):

self.ingredients.append(ingredient)

def remove\_ingredient(self, ingredient):

if ingredient in self.ingredients:

self.ingredients.remove(ingredient)

else:

print(f"{ingredient} is not in the ingredients list.")

def check\_matching\_ingredients(self, other\_recipe):

matching\_ingredients = set(self.ingredients) & set(other\_recipe.ingredients)

return list(matching\_ingredients)

def format\_recipe\_description(self):

formatted\_description = f"Recipe: {self.name}\n\nIngredients:\n"

for ingredient in self.ingredients:

formatted\_description += f"- {ingredient}\n"

formatted\_description += f"\nInstructions:\n{self.instructions}\n"

formatted\_description += f"Dietary Restrictions: {', '.join(self.dietary\_restrictions)}"

return formatted\_description

def compare\_on\_criteria(self, other\_recipe, criteria):

if criteria == 'name':

return self.name == other\_recipe.name

elif criteria == 'ingredients':

return set(self.ingredients) == set(other\_recipe.ingredients)

elif criteria == 'instructions':

return self.instructions == other\_recipe.instructions

elif criteria == 'dietary\_restrictions':

return set(self.dietary\_restrictions) == set(other\_recipe.dietary\_restrictions)

else:

return False # Invalid criteria

def \_\_eq\_\_(self, other):

return (

self.name == other.name and

self.ingredients == other.ingredients and

self.instructions == other.instructions and

self.dietary\_restrictions == other.dietary\_restrictions

)

def \_\_ne\_\_(self, other):

return not self.\_\_eq\_\_(other)

def \_\_add\_\_(self, ingredient):

new\_recipe = Recipe(self.name, self.ingredients[:], self.instructions, self.dietary\_restrictions[:])

new\_recipe.add\_ingredient(ingredient)

return new\_recipe

def \_\_sub\_\_(self, ingredient):

new\_recipe = Recipe(self.name, self.ingredients[:], self.instructions, self.dietary\_restrictions[:])

new\_recipe.remove\_ingredient(ingredient)

return new\_recipe

def \_\_lt\_\_(self, other):

return len(self.ingredients) < len(other.ingredients)

def \_\_gt\_\_(self, other):

return len(self.ingredients) > len(other.ingredients)

def \_\_le\_\_(self, other):

return len(self.ingredients) <= len(other.ingredients)

def \_\_ge\_\_(self, other):

return len(self.ingredients) >= len(other.ingredients)

@staticmethod

def find\_matching\_recipe(user\_ingredients, recipes):

matching\_recipes = []

for recipe in recipes:

if set(user\_ingredients) <= set(recipe.ingredients):

matching\_recipes.append(recipe)

return matching\_recipes

@staticmethod

def filter\_recipes\_by\_dietary\_restrictions(dietary\_restrictions, recipes):

filtered\_recipes = []

for recipe in recipes:

if set(dietary\_restrictions).issubset(set(recipe.dietary\_restrictions)):

filtered\_recipes.append(recipe)

return filtered\_recipes

recipe1 = Recipe(

"Pasta Carbonara",

["spaghetti", "eggs", "bacon", "parmesan", "black pepper"],

"Cook spaghetti. Mix eggs, parmesan, and cooked bacon. Add to spaghetti. Season with black pepper."

)

recipe2 = Recipe(

"Spaghetti Aglio e Olio",

["spaghetti", "garlic", "olive oil", "red pepper flakes", "parsley"],

"Cook spaghetti. Saute garlic and red pepper flakes in olive oil. Add to spaghetti. Garnish with parsley."

)

recipe1.dietary\_restrictions = ['Gluten-Free']

recipe2.dietary\_restrictions = ['Vegetarian']

recipe2.remove\_ingredient("parsley")

user\_provided\_ingredients = ["spaghetti", "garlic", "olive oil"]

matching\_recipes = Recipe.find\_matching\_recipe(user\_provided\_ingredients, [recipe1, recipe2])

if matching\_recipes:

print("Matching recipes found based on user ingredients:")

for match in matching\_recipes:

print(match.name)

else:

print("No matching recipes found based on user ingredients.")

user\_dietary\_restrictions = ['Gluten-Free']

filtered\_recipes = Recipe.filter\_recipes\_by\_dietary\_restrictions(user\_dietary\_restrictions, [recipe1, recipe2])

if filtered\_recipes:

print("\nFiltered recipes based on dietary restrictions:")

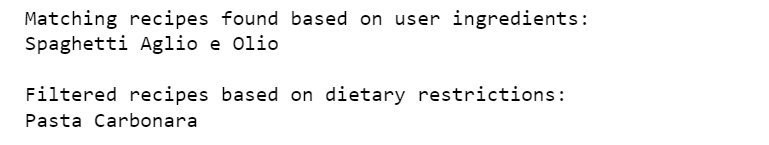
for recipe in filtered\_recipes:

print(recipe.name)

else:

print("\nNo recipes found that match the dietary restrictions.")

**OUPTUT:**

****

**QUESTION NO 2:**

class InventoryControlSystem:

def \_\_init\_\_(self):

self.inventory = {}

def add\_product(self, product\_id, product\_name, price, quantity):

if product\_id in self.inventory:

print("Product ID already exists. Use update\_product\_quantity to modify quantity.")

else:

self.inventory[product\_id] = {

'product\_name': product\_name,

'price': price,

'quantity': quantity

}

print(f"Product {product\_id} added to inventory.")

def update\_product\_quantity(self, product\_id, new\_quantity):

if product\_id in self.inventory:

self.inventory[product\_id]['quantity'] = new\_quantity

print(f"Quantity for Product {product\_id} updated.")

else:

print("Product ID not found in inventory.")

def remove\_product(self, product\_id):

if product\_id in self.inventory:

del self.inventory[product\_id]

print(f"Product {product\_id} removed from inventory.")

else:

print("Product ID not found in inventory.")

def check\_product\_availability(self, product\_id):

if product\_id in self.inventory:

available\_quantity = self.inventory[product\_id]['quantity']

print(f"Product {product\_id} is available. Available Quantity: {available\_quantity}.")

else:

print("Product ID not found in inventory.")

def display\_inventory(self):

print("Inventory Details:")

for product\_id, details in self.inventory.items():

print(f"Product ID: {product\_id}, Name: {details['product\_name']}, Price: {details['price']}, "

f"Quantity: {details['quantity']}")

def add\_new\_product(self, product\_id, product\_name, price, quantity):

if product\_id in self.inventory:

print("Product ID already exists. Use update\_product\_quantity to modify quantity.")

else:

self.add\_product(product\_id, product\_name, price, quantity)

def update\_product\_details(self, product\_id, product\_name=None, price=None, quantity=None):

if product\_id in self.inventory:

if product\_name:

self.inventory[product\_id]['product\_name'] = product\_name

if price:

self.inventory[product\_id]['price'] = price

if quantity:

self.inventory[product\_id]['quantity'] = quantity

print(f"Details for Product {product\_id} updated.")

else:

print("Product ID not found in inventory.")

inventory\_system = InventoryControlSystem()

inventory\_system.add\_product(1, "T-shirt", 15.99, 50)

inventory\_system.add\_product(2, "Jeans", 29.99, 30)

inventory\_system.add\_product(3, "Shoes", 49.99, 20)

inventory\_system.display\_inventory()

inventory\_system.check\_product\_availability(2)

inventory\_system.update\_product\_details(2, product\_name="New Jeans", price=39.99, quantity=40)

inventory\_system.display\_inventory()

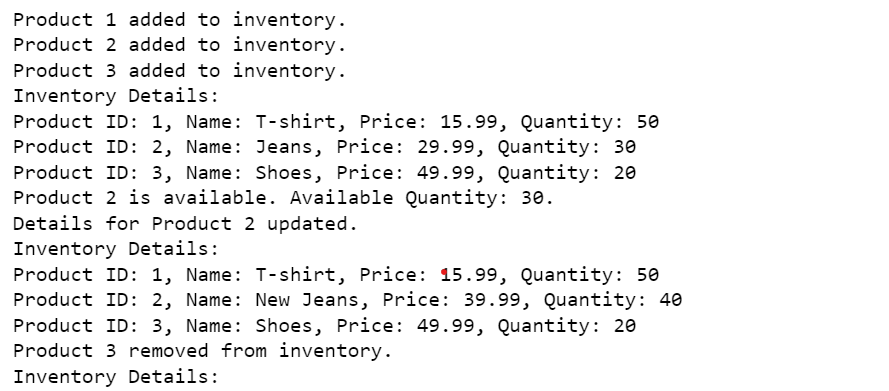
inventory\_system.remove\_product(3)

inventory\_system.display\_inventory()

inventory\_system.add\_new\_product(4, "Socks", 5.99, 100)

inventory\_system.display\_inventory()

**OUTPUT**



A black text with red mark

Description automatically generated

**QUESTION NO 3:**

import random

def generate\_random\_array(rows, cols):

return [[random.randint(1, 100) for \_ in range(cols)] for \_ in range(rows)]

def write\_array\_to\_file(array, filename):

with open(filename, 'w') as file:

for row in array:

file.write(' '.join(map(str, row)) + '\n')

def read\_array\_from\_file(filename):

with open(filename, 'r') as file:

return [[int(num) for num in line.split()] for line in file]

def print\_spiral\_order(matrix):

rows = len(matrix)

cols = len(matrix[0])

top, bottom, left, right = 0, rows - 1, 0, cols - 1

direction = 0

while top <= bottom and left <= right:

if direction == 0:

for i in range(left, right + 1):

print(matrix[top][i], end=" ")

top += 1

elif direction == 1:

for i in range(top, bottom + 1):

print(matrix[i][right], end=" ")

right -= 1

elif direction == 2:

for i in range(right, left - 1, -1):

print(matrix[bottom][i], end=" ")

bottom -= 1

elif direction == 3:

for i in range(bottom, top - 1, -1):

print(matrix[i][left], end=" ")

left += 1

direction = (direction + 1) % 4

random\_array = generate\_random\_array(5, 5)

write\_array\_to\_file(random\_array, 'random\_array.txt')

read\_array = read\_array\_from\_file('random\_array.txt')

print("Array read from file:")

for row in read\_array:

print(row)

print("\nSpiral Order:")

print\_spiral\_order(read\_array)

