#### Task 1: Sales Data Summary ####

# Step 1: Assign units sold for the two categories

category\_A\_units = 120 # Example: Replace with actual value

category\_B\_units = 80 # Example: Replace with actual value

# Step 2: Calculate metrics

total\_units\_sold = category\_a\_units + category\_b\_units

difference\_in\_units = abs(category\_a\_units - category\_b\_units) # Absolute difference

if category\_b\_units != 0:

ratio\_of\_units = category\_a\_units / category\_b\_units # Avoid division by zero

else:

ratio\_of\_units = "Undefined (Division by zero)"

# Step 3: Display results

print("Sales Data Summary:")

print(f"Category A Units Sold: {category\_a\_units}")

print(f"Category B Units Sold: {category\_b\_units}")

print(f"Total Units Sold: {total\_units\_sold}")

print(f"Difference in Units Sold: {difference\_in\_units}")

print(f"Ratio of Units Sold (A:B): {ratio\_of\_units}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Task 2: Customer Age Data ###

# List of customers with their names and ages

customers = [

{"name": "John Doe", "age": 30},

{"name": "Suresh Soman", "age": 25},

{"name": "James Bond", "age": 40}

]

# Generate messages for all customers

for customer in customers:

name = customer["name"]

age = customer["age"]

message = f"Dear {name}, at {age}, you’re eligible for our premium loyalty program."

print(message)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 3: Product List Management

# Initial list of product prices

product\_prices = [99.99, 49.99, 75.50, 150.00, 200.00, 125.00, 85.00]

# 1. Extract the highest and lowest prices

highest\_price = max(product\_prices)

lowest\_price = min(product\_prices)

print(f"Highest Price: ${highest\_price:.2f}")

print(f"Lowest Price: ${lowest\_price:.2f}")

# 2. Create a new list with mid-range products (excluding the highest and lowest)

mid\_range\_products = [price for price in product\_prices if price != highest\_price and price != lowest\_price]

print(f"Mid-Range Products: {mid\_range\_products}")

# 3. Add a new premium product price to the list

new\_premium\_price = 250.00

product\_prices.append(new\_premium\_price)

print(f"Updated Product List: {product\_prices}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 4: Inventory Lookup

# Step 1: Create a dictionary storing product details

product\_info = {

"product\_name": "Wireless Headphones",

"SKU": "WH-2024",

"price": 129.99,

"category": "Electronics"

}

# Step 2: Function to query product name and SKU

def lookup\_product(product\_details):

product\_name = product\_details.get("product\_name", "Unknown")

sku = product\_details.get("SKU", "Unknown")

return product\_name, sku

# Query by customer service representative

product\_name, sku = lookup\_product(product\_info)

# Step 3: Print the product name and SKU

print(f"Product Name: {product\_name}")

print(f"SKU: {sku}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 5: Stock Level Alert System

# Define the threshold for stock levels

stock\_threshold = 10

# Function to check stock level and print alerts

def check\_stock\_level(stock\_level):

if stock\_level < stock\_threshold:

print("Reorder Now: Stock level is critically low!")

else:

print("Stock is sufficient.")

# Example: Take stock level as input

try:

stock\_level = int(input("Enter the current stock level: "))

check\_stock\_level(stock\_level)

except ValueError:

print("Invalid input! Please enter a numeric value.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 6: Task 6: Sales Report Formatting

# Using For Loop Function

products\_sold = ["laptop", "smartphone", "headphones", "tablet", "smartwatch"]

print("Formatted Sales Report: ")

for product in products\_sold:

print(product.upper())

# Using For While Function

print("\nFormatted Sales Report: ")

index = 0

while index < len(products\_sold):

print(products\_sold[index].upper())

index += 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 7: Define a function to calculate the area of a store section

def calculate\_area(length, width):

"""

This function calculates the area of a rectangular section of the store.

:param length: Length of the section (float or int)

:param width: Width of the section (float or int)

:return: Area of the section (float or int)

"""

return length \* width

# Example store sections with their dimensions (length and width)

store\_sections = [

{"name": "Electronics Section", "length": 20, "width": 15},

{"name": "Clothing Section", "length": 25, "width": 18},

{"name": "Grocery Section", "length": 30, "width": 22},

{"name": "Toys Section", "length": 15, "width": 12},

]

for section in store\_sections:

name = section["name"]

length = section["length"]

width = section["width"]

# Calculate the area using the calculate\_area function

area = calculate\_area(length, width)

print(f"The area of the {name} is {area} square units.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 8: Customer Feedback Analysis

# Define a function to count vowels in a feedback message

def count\_vowels(feedback): #This function counts the number of vowels (a, e, i, o, u) in the given feedback.

"""

:param feedback: A string containing the customer feedback

:return: The count of vowels in the feedback

"""

# Define a set of vowels

vowels = "aeiouAEIOU"

# Initialize a counter for vowels

vowel\_count = 0

# Iterate through each character in the feedback

for char in feedback:

# Check if the character is a vowel

if char in vowels:

vowel\_count += 1 # Increment the counter if it's a vowel

return vowel\_count # Return the total vowel count

# Define a function to reverse the feedback message

def reverse\_feedback(feedback): #This function reverses the given feedback string.

"""

:param feedback: A string containing the customer feedback

:return: The reversed feedback string

"""

return feedback[::-1] # Use slicing to reverse the string

# Example customer feedback

customer\_feedback = "The service was excellent and the staff were friendly."

# Count the number of vowels in the feedback

vowel\_count = count\_vowels(customer\_feedback)

# Reverse the feedback message

reversed\_feedback = reverse\_feedback(customer\_feedback)

# Print the results

print(f"Original Feedback: {customer\_feedback}") # Display the original feedback

print(f"Number of Vowels: {vowel\_count}") # Display the number of vowels

print(f"Reversed Feedback: {reversed\_feedback}") # Display the reversed feedback

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 9: Price Filtering Tool

# Define a function to filter products based on a price threshold

def filter\_prices(products, price\_threshold): #This function filters products priced below a given threshold.

"""

:param products: A list of dictionaries containing product names and prices

:param price\_threshold: The minimum price threshold for eligibility

:return: A list of product names eligible for the discount

"""

# Use list comprehension to filter products based on the price threshold

eligible\_products = [product["name"] for product in products if product["price"] < price\_threshold]

return eligible\_products

# Example list of products with their prices

product\_list = [

{"name": "Laptop", "price": 1200},

{"name": "Smartphone", "price": 800},

{"name": "Headphones", "price": 150},

{"name": "Smartwatch", "price": 250},

{"name": "Tablet", "price": 500}

]

# Define the price threshold for the discount campaign

price\_threshold = 600

# Get the list of products eligible for the discount campaign

discounted\_products = filter\_prices(product\_list, price\_threshold)

# Print the results

print(f"Price Threshold: {price\_threshold}") # Display the price threshold

print(f"Eligible Products for Discount: {discounted\_products}") # Display the eligible products

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Task 10: Sales Log File Management

# Step 1: Create and write to the sales log file

def write\_sales\_log(file\_name, sales\_summary): #This function writes daily sales summaries to a log file.

"""

:param file\_name: Name of the file to write to

:param sales\_summary: A list of strings, each summarizing daily sales performance

"""

with open(file\_name, "w") as file:

# Write each sales summary line to the file

for line in sales\_summary:

file.write(line + "\n") # Add a newline character after each line

# Step 2: Read and print the content of the sales log file

def read\_sales\_log(file\_name): #This function reads and prints the content of a sales log file.

"""

:param file\_name: Name of the file to read from

"""

with open(file\_name, "r") as file:

# Read all lines from the file

content = file.readlines()

# Print each line to ensure data integrity

for line in content:

print(line.strip()) # Strip removes extra whitespace/newline characters

# File name for the sales log

sales\_log\_file = "sales\_log.txt"

# Example daily sales summaries

daily\_sales\_summary = [

"Total sales for the day: $12,500",

"Number of transactions: 320",

"Highest sale of the day: $1,200 by Product X"

]

# Write the sales summaries to the log file

write\_sales\_log(sales\_log\_file, daily\_sales\_summary)

# Read and verify the content of the sales log file

print("Sales Log Content:")

read\_sales\_log(sales\_log\_file)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 11: Daily Sales Average

# Step 1: Define a function to calculate the average sales

def calculate\_average\_sales(sales\_figures): #This function calculates the average daily sales from a list of sales figures.

"""

:param sales\_figures: A list of daily sales figures (float or int)

:return: The average sales (float)

"""

total\_sales = sum(sales\_figures) # Calculate the total sales

number\_of\_days = len(sales\_figures) # Calculate the number of days (length of the sales list)

average\_sales = total\_sales / number\_of\_days # Calculate the average sales

return average\_sales

# Step 2: List of sales figures for the past 7 days

weekly\_sales = [1250, 1300, 1500, 1100, 1350, 1400, 1450]

# Step 3: Calculate the average daily sales

average\_sales = calculate\_average\_sales(weekly\_sales)

# Step 4: Print the average sales

print("Weekly Sales Figures:", weekly\_sales) # Display the sales figures for reference

print(f"Average Daily Sales: ${average\_sales:.2f}") # Display the average sales rounded to 2 decimal places

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 12: Customer Segmentation

# Step 1: Define a function to categorize spending

def categorize\_spending(spending): #This function categorizes a customer's spending as Low, Medium, or High.

"""

:param spending: The total spending amount (float or int)

:return: A string indicating the spending category

"""

if spending < 500:

return "Low"

elif 500 <= spending < 1500:

return "Medium"

else:

return "High"

# Step 2: List of customer spending amounts

customer\_spending = [300, 1200, 700, 450, 2000, 900, 150]

# Step 3: Categorize each customer based on spending

categorized\_customers = []

for spending in customer\_spending:

category = categorize\_spending(spending) # Get the spending category

categorized\_customers.append({"Spending": spending, "Category": category}) # Store the result in a list

# Step 4: Print the categorized results

print("Customer Segmentation Results:")

for customer in categorized\_customers:

print(f"Spending: ${customer['Spending']}, Category: {customer['Category']}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 13: Discount Calculation

# Step 1: Define a function to calculate the final price after applying a discount

def calculate\_discounted\_price(original\_price, discount\_percentage): # This function calculates the final price of a product after applying a discount.

"""

:param original\_price: The original price of the product (float or int)

:param discount\_percentage: The discount percentage to be applied (float or int)

:return: The final price after applying the discount (float)

"""

discount\_amount = (original\_price \* discount\_percentage) / 100 # Calculate the discount amount

final\_price = original\_price - discount\_amount # Subtract the discount amount from the original price to get the final price

return final\_price

# Step 2: List of products with their original prices and discounts

products = [

{"name": "Laptop", "original\_price": 1200, "discount\_percentage": 10},

{"name": "Smartphone", "original\_price": 800, "discount\_percentage": 15},

{"name": "Headphones", "original\_price": 150, "discount\_percentage": 5},

{"name": "Smartwatch", "original\_price": 250, "discount\_percentage": 20},

{"name": "Tablet", "original\_price": 500, "discount\_percentage": 12}

]

# Step 3: Calculate the final prices for all products

final\_prices = []

for product in products:

final\_price = calculate\_discounted\_price(product["original\_price"], product["discount\_percentage"]) # Calculate the discounted price for each product

final\_prices.append({"name": product["name"], "final\_price": final\_price}) # Store the results in a list along with product details

# Step 4: Print the results

print("Discounted Prices:")

for item in final\_prices:

print(f"Product: {item['name']}, Final Price: ${item['final\_price']:.2f}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 14: Customer Feedback Sentiment Analysis

# Step 1: Define a function to analyze sentiment based on keywords

def analyze\_sentiment(feedback): #This function checks the customer feedback for positive or negative words.

"""

:param feedback: A string containing the customer feedback

:return: A string indicating whether the feedback is 'Positive' or 'Negative'

"""

# Define lists of positive and negative words

positive\_words = ["good", "happy", "excellent", "great", "satisfied"]

negative\_words = ["bad", "disappointed", "poor", "angry", "unhappy"]

feedback = feedback.lower() # Convert feedback to lowercase to make the comparison case-insensitive

# Check if any positive or negative word is in the feedback

for word in positive\_words:

if word in feedback:

return "Positive" # If any positive word is found, return "Positive"

for word in negative\_words:

if word in feedback:

return "Negative" # If any negative word is found, return "Negative"

return "Neutral" # If no positive or negative words are found, return "Neutral"

# Step 2: Example customer feedback

customer\_feedback = "I am very happy with the service, it was great!"

# Step 3: Analyze the sentiment of the feedback

sentiment = analyze\_sentiment(customer\_feedback)

# Step 4: Print the sentiment result

print(f"Customer Feedback Sentiment: {sentiment}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 15: Employee Salary Increment Calculator

# Step 1: Define a function to calculate the salary increment

def calculate\_salary\_increment(salary, rating): #This function calculates the salary increment based on the performance rating.

"""

:param salary: The current salary of the employee

:param rating: The performance rating (1 to 5)

:return: The updated salary after applying the increment

"""

# Define increment percentages based on ratings

if rating == 5:

increment\_percentage = 20 # 20% increment for excellent performance

elif rating == 4:

increment\_percentage = 15 # 15% increment for good performance

elif rating == 3:

increment\_percentage = 10 # 10% increment for average performance

elif rating == 2:

increment\_percentage = 5 # 5% increment for below-average performance

else:

increment\_percentage = 0 # No increment for poor performance

increment\_amount = (salary \* increment\_percentage) / 100 # Calculate the increment amount

updated\_salary = salary + increment\_amount # Calculate the updated salary

return updated\_salary

# Step 2: Dictionary storing employee names and their performance ratings

employees = {

"Alice": {"salary": 50000, "rating": 5},

"Bob": {"salary": 40000, "rating": 3},

"Charlie": {"salary": 60000, "rating": 4},

"David": {"salary": 35000, "rating": 2},

"Eve": {"salary": 55000, "rating": 1}

}

# Step 3: Calculate the updated salary for each employee

for employee, details in employees.items():

updated\_salary = calculate\_salary\_increment(details["salary"], details["rating"]) # Calculate the updated salary based on rating

# Print the updated salary for the employee

print(f"Employee: {employee}, Original Salary: ${details['salary']}, Rating: {details['rating']}, Updated Salary: ${updated\_salary:.2f}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 16: Monthly Sales Report Generator

import random

# Step 1: Generate a list of daily sales for the month (30 days)

# Let's assume daily sales values range from 100 to 1000 (in dollars or any currency unit)

daily\_sales = [random.randint(100, 1000) for \_ in range(30)]

# Step 2: Calculate the total sales for the month

total\_sales = sum(daily\_sales)

# Step 3: Calculate the average sales for the month

average\_sales = total\_sales / len(daily\_sales)

# Step 4: Write the statistics to a text file named 'monthly\_report.txt'

with open('monthly\_report.txt', 'w') as file:

file.write("Monthly Sales Report\n")

file.write("--------------------\n")

file.write(f"Total Sales: ${total\_sales}\n")

file.write(f"Average Daily Sales: ${average\_sales:.2f}\n")

# Print confirmation that the file has been created

print("Monthly sales report has been generated and saved to 'monthly\_report.txt'.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 17: Stock Replenishment Planning

# Step 1: List of products and their current stock levels

products = [

{"name": "Product A", "stock": 50},

{"name": "Product B", "stock": 20},

{"name": "Product C", "stock": 5},

{"name": "Product D", "stock": 100},

{"name": "Product E", "stock": 0},

]

# Step 2: Define the threshold for each product

thresholds = {

"Product A": 30,

"Product B": 25,

"Product C": 10,

"Product D": 50,

"Product E": 5,

}

# Step 3: Compare current stock levels with the threshold

products\_to\_replenish = []

for product in products:

product\_name = product["name"]

current\_stock = product["stock"]

threshold = thresholds.get(product\_name, 0) # Default threshold of 0 if not specified

if current\_stock <= threshold:

products\_to\_replenish.append(product\_name)

# Step 4: Print the products that need to be reordered

if products\_to\_replenish:

print("The following products need replenishment:")

for product in products\_to\_replenish:

print(product)

else:

print("No products need replenishment at the moment.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Task 18: Data Cleaning Utility

# Step 1: Input list of customer names with extra spaces and inconsistent capitalization

customer\_names = [

" JOHN DOE ",

" jane doe",

" mARY SMITH ",

" bOB Johnson ",

"alice WILLIAMS"

]

# Step 2: Clean the names

cleaned\_names = []

for name in customer\_names: # Trim leading and trailing spaces and standardize capitalization

cleaned\_name = name.strip().title() # .strip() removes extra spaces, .title() capitalizes correctly

cleaned\_names.append(cleaned\_name)

# Step 3: Print the cleaned names

print("Cleaned Customer Names:")

for name in cleaned\_names:

print(name)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 19: Simple Sales Forecasting

# Step 1: Input sales data for the last 3 months

sales\_last\_three\_months = [10000, 6000, 14000] # Example sales figures for the last 3 months

# Step 2: Calculate the average sales for the last 3 months

average\_sales = sum(sales\_last\_three\_months) / len(sales\_last\_three\_months)

# Step 3: Forecast next month's sales

forecasted\_sales = average\_sales

# Step 4: Print the forecasted sales for budget planning

print(f"Forecasted Sales for Next Month: ${forecasted\_sales:.2f}")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Task 20: Customer Loyalty Points Calculator

# Step 1: List of customers and their total purchase amount

customers = [

{"name": "Suresh Soman", "total\_purchase": 120},

{"name": "Sunil Soman", "total\_purchase": 450},

{"name": "Jimmy John", "total\_purchase": 25},

{"name": "Sunny Dhaman", "total\_purchase": 600},

{"name": "Rajat Gupta", "total\_purchase": 200}

]

# Step 2: Define the tiered system for loyalty points

def calculate\_loyalty\_points(total\_purchase):

if total\_purchase < 100:

return total\_purchase \* 1 # 1 point per dollar

elif 100 <= total\_purchase < 500:

return total\_purchase \* 1.5 # 1.5 points per dollar

else:

return total\_purchase \* 2 # 2 points per dollar

# Step 3: Calculate and print the loyalty points for each customer

for customer in customers:

name = customer["name"]

total\_purchase = customer["total\_purchase"]

loyalty\_points = calculate\_loyalty\_points(total\_purchase)

print(f"Customer: {name}")

print(f"Total Purchase: ${total\_purchase}")

print(f"Loyalty Points: {loyalty\_points:.2f}")

print("-" \* 30)