1. Accept a date and the number of days to add it and print the resultant date.

from datetime import datetime, timedelta

# Input from the user

beginDate = input("Enter the date: ")

no0fDays = int(input("Enter the number of days: "))

# Making it in Date format after getting input as year, month and day

aDate = datetime.strptime (beginDate, "%Y-%m-%d")

# Adding the date with number of days

resDate = aDate + timedelta (days=no0fDays)

print("The resultant date is: ", resDate)

Output:

Enter the date: 2025-05-10

Enter the number of days: 5

The resultant date is: 2025-05-15 00:00:00

2. Accept a 5-digit decimal number as input and display the sum of the digits as output.

def sumOfDigits(number):

result = 0

for i in str(number):

if i.isdigit():

result += int(i)

return result

# Input from the user

decimalNumber = float(input("Enter a 5-digit decimal number: "))

# Converting the decimal number to string

number = str(decimalNumber)

# Display the result

print("The sum of the digits is:", sumOfDigits(number))

Output:

Enter a 5-digit decimal number: 123.45

The sum of the digits is: 15

3. An employee is considered for on-site depending on these conditions

(i) An employee Should have Passport

(ii) Communication should be good

(iii) His training feedback should be good

(iv) Should be at-least 2years experienced.

(v) Age should be greater than or equal to 23. Using above conditions, check if an employee is eligible to go to on-site or not.

# Input from the user

hasPassport = input("Do you have a passport? (Yes/No): ")

hasGoodCommunication = input("Do you have a good communication? (Yes/No): ")

hasGoodTrainingFeedback = input("Do you have a good training feedback? (Yes/No): ")

yearsOfExperience = int(input("How many years of work experience do you have? (An Integer): "))

age = int(input("What is your age? (An Integer): "))

# Considering an employee for on-site work or not

if hasPassport == "Yes":

if hasGoodCommunication == "Yes":

if hasGoodTrainingFeedback == "Yes":

if yearsOfExperience >= 2:

if age >= 23:

print("Employee is eligible for on-site work.")

else:

print("Employee is not eligible for on-site work.")

Output:

Do you have a passport? (Yes/No): Yes

Do you have a good communication? (Yes/No): Yes

Do you have a good training feedback? (Yes/No): Yes

How many years of work experience do you have? (An Integer): 3

What is your age? (An Integer): 25

Employee is eligible for on-site work.

4. Calculate electricity bill for following constraints.

(i) If units exceed 1000, then charge Rs. 10/- per unit.

(ii) If units exceed 500, then charge Rs.5/- per unit.

(iii) If units exceed 200, then charge Rs.2/- per unit.

(iv) In other cases, charge Rs.1/- per unit.

# Input from the user

unitsOfElectricityConsumed = int(input("What is the amount of electricity consumed in units? (An Integer): "))

# Calculating the electricity bill

if unitsOfElectricityConsumed > 1000:

costOfElectricity = 10 \* unitsOfElectricityConsumed

elif unitsOfElectricityConsumed > 500:

costOfElectricity = 5 \* unitsOfElectricityConsumed

elif unitsOfElectricityConsumed > 200:

costOfElectricity = 2 \* unitsOfElectricityConsumed

else:

costOfElectricity = 1 \* unitsOfElectricityConsumed

# Display the result

print("The electricity bill is Rs.", costOfElectricity, "/-")

Output:

What is the amount of electricity consumed in units? (An Integer): 12

The electricity bill is Rs. 12 /-

5. Implement the following logic to arrive at the loan eligibility for an employee:

(a) For an unmarried permanent job holder, if the service has been for 30 years or more, the loan amount is Rs.50,000; otherwise, the loan amount is Rs.25,000

(b) For a married permanent job holder, if the service has been for 30 years or more, the loan amount is Rs.60,000; otherwise, the loan amount is Rs.35,000

(c) For temporary job holder, the loan amount is Rs. 10,000 Test the application using the below given test cases

(a) a married temporary job holder

(b) an unmarried temporary job holder

(c) a temporary job holder with 30 years of service

(d) a married permanent job holder with 30 years of service

(e) an unmarried permanent job holder with 30 years' service

(f) a married permanent job holder with 25 years' service.

# Input from the user

typeOfJobHolder = input("What type of job do you have? (Permanent/Temporary): ").strip().capitalize()

maritalStatus = input("What is your marital status? (Married/Unmarried): ").strip().capitalize()

yearsOfExperience = int(input("How many years of work experience do you have? (An Integer): "))

# Considering an employee for loan eligibility

if typeOfJobHolder == "Temporary":

print("Employee is eligible for a loan amount of Rs. 10,000.")

elif typeOfJobHolder == "Permanent":

if maritalStatus == "Married":

if yearsOfExperience >= 30:

print("Employee is eligible for a loan amount of Rs. 60,000.")

else:

print("Employee is eligible for a loan amount of Rs. 35,000.")

elif maritalStatus == "Unmarried":

if yearsOfExperience >= 30:

print("Employee is eligible for a loan amount of Rs. 50,000.")

else:

print("Employee is eligible for a loan amount of Rs. 25,000.")

else:

print("Invalid marital status entered.")

else:

print("Invalid job type entered.")

1. Output:

What type of job do you have? (Permanent/Temporary): Temporary

What is your marital status? (Married/Unmarried): Married

How many years of work experience do you have? (An Integer): 6

Employee is eligible for a loan amount of Rs. 10,000.

1. Output:

What type of job do you have? (Permanent/Temporary): Temporary

What is your marital status? (Married/Unmarried): Unmarried

How many years of work experience do you have? (An Integer): 10

Employee is eligible for a loan amount of Rs. 10,000.

1. Output:

What type of job do you have? (Permanent/Temporary): Temporary

What is your marital status? (Married/Unmarried): Unmarried

How many years of work experience do you have? (An Integer): 30

Employee is eligible for a loan amount of Rs. 10,000.

1. Output:

What type of job do you have? (Permanent/Temporary): Permanent

What is your marital status? (Married/Unmarried): Married

How many years of work experience do you have? (An Integer): 30

Employee is eligible for a loan amount of Rs. 60,000.

1. Output:

What type of job do you have? (Permanent/Temporary): Permanent

What is your marital status? (Married/Unmarried): Unmarried

How many years of work experience do you have? (An Integer): 30

Employee is eligible for a loan amount of Rs. 50,000.

1. Output:

What type of job do you have? (Permanent/Temporary): Permanent

What is your marital status? (Married/Unmarried): Married

How many years of work experience do you have? (An Integer): 30

Employee is eligible for a loan amount of Rs. 35,000.

6. If in a year January 1 st is Sunday, then list all the dates of Sunday in that month.

from datetime import date, timedelta

# Given starting date is January 1st, Sunday

startDate = date(2025, 1, 1)

# Create an empty list of all Sundays in January

sundays = []

# Initialization for the while loop

currentDate = startDate

# Add 7 days repeatedly until we leave January

while currentDate.month == 1:

sundays.append(currentDate)

currentDate += timedelta(days=7)

# Display the results

print("The Sundays in January are: ")

for i in sundays:

print(i)

Output:

The Sundays in January are:

2025-01-01

2025-01-08

2025-01-15

2025-01-22

2025-01-29

7. Print Series

1/2, 2/5, 3/10, 4/17, 5/26, 6/37,...

0, 3, 8, 15, 24, 35, 48, 63,...

0, 2, 7, 14, 25, 38,…

# Number of terms in series 1

series1N = 6

for n in range(1, series1N + 1):

numerator = n

denominator = n\*\*2 + 1

print(f"{numerator}/{denominator}", end=", ")

print()

# Number of terms in series 2

series2N = 8

for n in range(1, series2N + 1):

value = n\*\*2 - 1

print(value, end=", ")

# Number of terms in series 3

series3N = 6

Output:

1/2, 2/5, 3/10, 4/17, 5/26, 6/37,

0, 3, 8, 15, 24, 35, 48, 63,

8. Find the sum of the elements of the array.

import array

# Define an array

arr = array.array('i', [])

# Input from the user

no\_of\_elements = int(input("Enter the number of elements to be inserted in the array: "))

for i in range(no\_of\_elements):

arr.append(int(input("Enter element {}: ".format(i + 1))))

# Recursive function to find sum of elements

def sum\_of\_elements(arr):

if len(arr) == 1:

return arr[0]

else:

return arr[0] + sum\_of\_elements(arr[1:])

# Display the result

print("The sum of elements is", sum\_of\_elements(arr))

Output:

Enter the number of elements to be inserted in the array: 5

Enter element 1: 4

Enter element 2: 6

Enter element 3: 9

Enter element 4: 3

Enter element 5: 8

The sum of elements is 30

9. Read a sentence. Find a word and replace it with other word. (Note: Both words should be same in length)

# Original sentence

sentence = "I am smart and amazing."

print("Original sentence:", sentence)

# Input from the user

changeWord = input("Enter the word you want to change in the above sentence: ")

replaceWord = input("Enter the word you want to replace it with: ")

# Check if both words are of the same length

if len(changeWord) != len(replaceWord):

print("Both words should be same in length. Try again!")

else:

# Replace the word in the sentence

new\_sentence = sentence.replace(changeWord, replaceWord)

# Display the new changed sentence

print("The new sentence is:", new\_sentence)

Output:

Original sentence: I am smart and amazing.

Enter the word you want to change in the above sentence: smart

Enter the word you want to replace it with: brave

The new sentence is: I am brave and amazing.

10. Create a function that takes a sentence as input and does the following validation

(a) Length of the sentence must me minimum 10

(b) Sentence should have at-least one space.

(c) Sentence should not have dot in between.

(d) Should not start and end with spaces

(e) Must end with dot

def validate\_sentence(sentence):

if len(sentence) < 10:

return "Sentence must be at least 10 characters long."

if ' ' not in sentence:

return "Sentence must contain at least one space."

if '.' in sentence[:-1]: # Check for dot not at the end

return "Dot is not allowed in the middle of the sentence."

if sentence.startswith(" "):

return "Sentence should not start with space."

if sentence.endswith(" "):

return "Sentence should not end with space."

if not sentence.endswith("."):

return "Sentence must end with a dot."

return "The sentence is valid."

# Input from the user

sentence = input("Enter a sentence: ")

print(validate\_sentence(sentence))

Output:

Enter a sentence: I am smart and amazing.

The sentence is valid.

11. Write a function validatelPaddress that accepts an ipAddress (string) as an input and dons the following validations

(a) Length should be minimum 7 and maximum 15

(b) Should not start and end with dot

(c) Should have exactly 3 dots

(d) Dots should not be in consecutive positions.

(e) Should not permit alphabets and special characters

def validateIPaddress(ip):

if len(ip) < 7:

return "IP address length should be minimum 7."

if len(ip) > 15:

return "IP address length should be maximum 15."

if ip.startswith("."):

return "IP address should not start with a dot."

if ip.endswith("."):

return "IP address should not end with a dot."

if ip.count(".") != 3:

return "IP address should have exactly 3 dots."

if ip.find('..') != -1:

return "Dots should not be in consecutive positions."

parts = ip.split(".")

for part in parts:

if part == "":

continue

if part.isdigit() == False:

return "IP address must contain only digits and dots."

return "The IP address is valid."

# Input from the user

ipAddress = input("Enter an IP address: ")

print(validateIPaddress(ipAddress))

Output:

Enter an IP address: 255.255.255.255

The IP address is valid.

12. Provide logic to validate a given a password whether the password is valid or not

(a) Length should be minimum 10 characters and maximum 50 characters

(b) Should not start and end with spaces

(c) Should not contain any special character other than underscore (\_)

import string

def validate\_password(password):

if len(password) < 10:

return "The password should be minimum 10 characters."

if len(password) > 50:

return "The password should be maximum 50 characters."

if password.startswith(" "):

return "The password should not start with space."

if password.endswith(" "):

return "The password should not end with space."

allowed\_chars = string.ascii\_letters + string.digits + '\_'

for i in password:

if allowed\_chars.find(i) == -1:

return "Only letters, digits, and underscore (\_) are allowed."

return "The password is valid."

# Input from the user

password = input("Enter a password for validation: ")

print(validate\_password(password))

Output:

Enter a password for validation: abcde\_12345

The password is valid.

Loan Approval System

A person qualifies for a loan if:

a)Has a stable job

b)Has a credit score above 750

c)Earns at least ₹50,000 per month

d)Has no previous loan defaults

e)Has a valid income proof

Write a program to check loan eligibility.

stableJob = input("Do you have a stable job? (True/False): ")

creditScore = int(input("Enter your credit score: "))

salary = int(input("Enter your salary in ₹ per month: "))

loanDefaults = input("Do you have any previous loan defaults? (True/False): ")

validIncomeProof = input("Do you have a valid income proof? (True/False): ")

if stable\_job == "True":

if credit\_score >= 750:

if salary >= 50000:

if loan\_defaults == "False":

if valid\_income\_proof == "True":

print("Qualified for a loan")

else:

print("Not qualified for a loan")

Scholarship Eligibility

A student qualifies for a scholarship if:

a)Has at least 85% attendance

b)Maintains a CGPA above 8.0

c)Has no disciplinary actions

d)Has financial need (income < ₹5,00,000)

e)Has received positive teacher recommendations

Write a program to check scholarship eligibility.

attendance = float(input("Enter the attendance percentage of the student: "))

cgpa = float(input("Enter the CGPA of the student: "))

disciplinaryAction = input("Does the student have any disciplinary actions? (Yes/No): ") == "No"

income = float(input("Enter the family income of the student in ₹: "))

teacherRecommendation = input("Does the student have positive teacher recommendations? (Yes/No): ") == "Yes"

if attendance >= 85:

if cgpa > 8.0:

if disciplinaryAction:

if income < 500000:

if teacherRecommendation:

print("Eligible for scholarship")

else:

print("Not eligible for scholarship")

Car Insurance Premium Discount

A driver qualifies for an insurance discount if:

a)Has no accident history

b)Has at least 5 years of driving experience

c)Has a valid insurance policy

d)Has a clean traffic record (no major violations)

e)Renews the policy before the due date

Write a program to determine if a driver gets a discount.

accidentHistory = input("Do you have any accident history? (Yes/No): ") == "No"

drivingExperience = int(input("Enter your years of driving experience: "))

validPolicy = input("Do you have a valid policy? (Yes/No): ") == "Yes"

trafficRecord = input("Do you have a clean traffic record? (Yes/No): ") == "Yes"

policyRenewal = input("Do you have renewed policy before due date? (Yes/No): ") == "Yes"

if accidentHistory:

if drivingExperience >= 5:

if validPolicy:

if trafficRecord:

if policyRenewal:

print("Eligible for insurance discount")

else:

print("Not eligible for insurance discount")

Job Promotion Check

An employee is eligible for promotion if:

a)Has completed at least 3 years in the company

b)Has a performance rating of 4.0 or higher

c)Has attended leadership training

d)Has taken on additional responsibilities

e)Has received positive peer feedback

Write a program to check job promotion eligibility.

yearsInCompany = int(input("Enter years in the company: "))

performanceRating = float(input("Enter performance rating (out of 5): "))

leadershipTraining = input("Completed leadership training? (Yes/No): ") == "Yes"

additionalResponsibilities = input("Taken additional responsibilities? (Yes/No): ") == "Yes"

peerFeedback = input("Positive peer feedback? (Yes/No): ") == "Yes"

if yearsInCompany >= 3:

if performanceRating >= 4.0:

if leadershipTraining:

if additionalResponsibilities:

if peerFeedback:

print("Eligible for promotion!")

else:

print("Not eligible for promotion.")

Flight Ticket Discount

A person gets a discount on a flight ticket if:

a)Is a senior citizen (age >= 60) OR a student (age <= 21)

b)Has a valid student ID (if student)

c)Has booked the ticket at least 30 days in advance

d)Is a frequent flyer member

e)Has a special promo code

Write a Python program to determine if a passenger gets a discount.

age = int(input("Enter your age: "))

if age <= 21:

validStudentId = input("Do you have a valid student ID? (Yes/No): ") == "Yes"

else:

validStudentId = True

daysInAdvance = int(input("Enter the number of days you booked the ticket in advance: "))

frequentFlyer = input("Are you a frequent flyer member? (Yes/No): ") == "Yes"

promoCode = input("Do you have a special promo code? (Yes/No): ") == "Yes"

if (age >= 60 or (age <= 21 and validStudentId)) and daysInAdvance >= 30 and frequentFlyer and promoCode:

print("Eligible for flight ticket discount")

else:

print("Not eligible for flight ticket discount")

Fitness Program Eligibility

A gym allows membership if:

a)The person is between 18 and 60 years old

b)Their BMI is between 18.5 and 24.9

c)They have no serious health issues

d)They complete a fitness assessment test

e)They sign a health declaration form

Write a program to check gym membership eligibility.

age = int(input("Enter your age: "))

bmi = float(input("Enter your BMI: "))

health\_issues = input("Do you have any serious health issues? (Yes/No): ") == "No"

fitness\_test = input("Have you completed the fitness assessment test? (Yes/No): ") == "Yes"

health\_declaration = input("Have you signed the health declaration form? (Yes/No): ") == "Yes"

if 18 <= age <= 60 and 18.5 <= bmi <= 24.9 and health\_issues and fitness\_test and health\_declaration:

print("Eligible for gym membership")

else:

print("Not eligible for gym membership")

University Admission Criteria

A student qualifies for admission if:

a)Their entrance exam score is above 85%

b)They have a strong personal statement

c)They have at least 2 recommendation letters

d)Their extracurricular activities are notable

e)They have cleared the interview round

Write a program to check university admission eligibility.

exam\_score = float(input("Enter your entrance exam score in %: "))

personal\_statement = input("Do you have a strong personal statement? (Yes/No): ") == "Yes"

recommendation\_letters = int(input("Enter the number of recommendation letters: "))

extracurricular\_activities = input("Do you have notable extracurricular activities? (Yes/No): ") == "Yes"

interview\_clearance = input("Have you cleared the interview round? (Yes/No): ") == "Yes"

if exam\_score > 85 and personal\_statement and recommendation\_letters >= 2 and extracurricular\_activities and interview\_clearance:

print("Eligible for university admission")

else:

print("Not eligible for university admission")

Vehicle Registration Eligibility

A vehicle can be registered if:

a)It passes the pollution test

b)The owner has valid insurance

c)The vehicle is not older than 15 years

d)It has all necessary documents (RC, PUC, etc.)

e)The owner has paid all pending traffic fines

Write a program to verify if a vehicle can be registered.

pollution\_test = input("Did the vehicle pass the pollution test? (Yes/No): ") == "Yes"

valid\_insurance = input("Does the vehicle have valid insurance? (Yes/No): ") == "Yes"

vehicle\_age = int(input("Enter the age of the vehicle in years: "))

necessary\_docs = input("Do you have all necessary documents (RC, PUC, etc.)? (Yes/No): ") == "Yes"

pending\_fines = input("Have all pending traffic fines been paid? (yes/no): ") == "yes"

if pollution\_test and valid\_insurance and vehicle\_age <= 15 and necessary\_docs and pending\_fines:

print("Vehicle is eligible for registration")

else:

print("Vehicle is not eligible for registration")

Business Loan Approval

A business qualifies for a loan if:

a)Has been operational for more than 3 years

b)Has a profit margin of at least 10%

c)Has all legal documentation (GST, PAN, etc.)

d)Has no outstanding loan defaults

e)Can provide collateral or a guarantor

Write a program to determine business loan eligibility.

years\_operational = int(input("Enter the number of years the business has been operational: "))

profit\_margin = float(input("Enter the profit margin (percentage): "))

legal\_docs = input("Does the business have all legal documentation (GST, PAN, etc.)? (Yes/No): ") == "Yes"

loan\_defaults = input("Are there any outstanding loan defaults? (Yes/No): ") == "No"

collateral\_or\_guarantor = input("Can you provide collateral or a guarantor? (Yes/No): ") == "Yes"

if years\_operational > 3 and profit\_margin >= 10 and legal\_docs and loan\_defaults and collateral\_or\_guarantor:

print("Business is eligible for loan approval")

else:

print("Business is not eligible for loan approval")

Hotel Room Booking Criteria

A person can book a hotel room if:

a)Is at least 18 years old

b)Has a valid ID proof

c)Provides a security deposit (if required)

d)Has a confirmed reservation

e)Is not on the hotel's blacklist

Write a program to check if a person can book a hotel room.

age = int(input("Enter your age: "))

id\_proof = input("Do you have a valid ID proof? (Yes/No): ") == "Yes"

security\_deposit = input("Have you provided a security deposit (if required)? (Yes/No): ") == "Yes"

reservation = input("Do you have a confirmed reservation? (Yes/No): ") == "Yes"

blacklist = input("Are you on the hotel's blacklist? (Yes/No): ") == "No"

if age >= 18 and id\_proof and security\_deposit and reservation and blacklist:

print("Eligible to book a hotel room")

else:

print("Not eligible to book a hotel room")

Online Course Access

A user gets access to an online course if:

a)Has paid the full course fee

b)Has accepted the terms and conditions

c)Has completed account verification

d)Has a stable internet connection

e)Has a registered email address

Write a program to verify course access.

course\_fee\_paid = input("Have you paid the full course fee? (Yes/No): ") == "Yes"

terms\_accepted = input("Have you accepted the terms and conditions? (Yes/No): ") == "Yes"

account\_verified = input("Is your account verified? (Yes/No): ") == "Yes"

internet\_connection = input("Do you have a stable internet connection? (Yes/No): ") == "Yes"

registered\_email = input("Do you have a registered email address? (Yes/No): ") == "Yes"

if course\_fee\_paid and terms\_accepted and account\_verified and internet\_connection and registered\_email:

print("Access granted to the online course")

else:

print("Access denied to the online course")

Club Membership Eligibility

A person qualifies for club membership if:

a)Is above 25 years old

b)Earns more than ₹75,000 per month

c)Has a strong financial background

d)Has at least two references from existing members

e)Has no previous membership bans

Write a program to check if a person qualifies for membership.

age = int(input("Enter your age: "))

income = int(input("Enter your monthly income in ₹: "))

financial\_background = input("Do you have a strong financial background? (Yes/No): ") == "Yes"

references = int(input("Enter the number of references from existing members: "))

previous\_bans = input("Do you have any previous membership bans? (Yes/No): ") == "No"

if age > 25 and income > 75000 and financial\_background and references >= 2 and previous\_bans:

print("Eligible for club membership")

else:

print("Not eligible for club membership")

Festival Bonus for Employees

An employee gets a festival bonus if:

a)Has been with the company for at least 5 years

b)Has a performance rating above 4.5

c)Has not taken more than 10 days of leave in the year

d)Has achieved their annual targets

e)Is not involved in any disciplinary action

Write a program to check festival bonus eligibility.

years\_in\_company = int(input("Enter your years in the company: "))

performance\_rating = float(input("Enter your performance rating (out of 5): "))

leave\_days = int(input("Enter the number of leave days taken this year: "))

annual\_targets = input("Have you achieved your annual targets? (Yes/No): ") == "Yes"

disciplinary\_action = input("Are you involved in any disciplinary action? (Yes/No): ") == "No"

if years\_in\_company >= 5 and performance\_rating > 4.5 and leave\_days <= 10 and annual\_targets and disciplinary\_action:

print("Eligible for festival bonus")

else:

print("Not eligible for festival bonus")

Online Shopping Discount Eligibility

A customer gets a discount if:

a)Their total purchase is above ₹5,000

b)They are a premium member

c)They have a valid coupon code

d)They are shopping during a special sale event

e)They are a first-time customer

Write a program to check if a customer gets a discount.

total\_purchase = float(input("Enter your total purchase amount in ₹: "))

premium\_member = input("Are you a premium member? (Yes/No): ") == "Yes"

coupon\_code = input("Do you have a valid coupon code? (Yes/No): ") == "Yes"

sale\_event = input("Are you shopping during a special sale event? (Yes/No): ") == "Yes"

first\_time\_customer = input("Are you a first-time customer? (Yes/No): ") == "Yes"

if total\_purchase > 5000 and premium\_member and coupon\_code and sale\_event and first\_time\_customer:

print("Eligible for online shopping discount")

else:

print("Not eligible for online shopping discount")

Citizenship Eligibility Check

A person qualifies for citizenship if:

a)Has lived in the country for more than 10 years

b)Has no criminal record

c)Has a valid residence permit

d)Can speak the national language

e)Has passed the citizenship test

Write a program to check citizenship eligibility.

years\_lived = int(input("Enter the number of years you have lived in the country: "))

criminal\_record = input("Do you have any criminal record? (Yes/No): ") == "No"

residence\_permit = input("Do you have a valid residence permit? (Yes/No): ") == "Yes"

national\_language = input("Can you speak the national language? (Yes/No): ") == "Yes"

citizenship\_test = input("Have you passed the citizenship test? (Yes/No): ") == "Yes"

if years\_lived > 10 and criminal\_record and residence\_permit and national\_language and citizenship\_test:

print("Eligible for citizenship")

else:

print("Not eligible for citizenship")

Sports Team Selection

A player qualifies for selection if:

a)Has a fitness score above 80

b)Has a trial performance above 85%

c)Has at least 3 years of competitive experience

d)Has a clean disciplinary record

e)Has coach recommendations

Write a program to check if a player is selected.

fitness\_score = int(input("Enter your fitness score: "))

trial\_performance = float(input("Enter your trial performance percentage: "))

competitive\_experience = int(input("Enter your years of competitive experience: "))

disciplinary\_record = input("Do you have a clean disciplinary record? (Yes/No): ") == "Yes"

coach\_recommendations = input("Do you have coach recommendations? (Yes/No): ") == "Yes"

if fitness\_score > 80 and trial\_performance > 85 and competitive\_experience >= 3 and disciplinary\_record and coach\_recommendations:

print("Eligible for sports team selection")

else:

print("Not eligible for sports team selection")

Government Job Eligibility

A candidate qualifies for a government job if:

a)Has cleared the entrance exam

b)Has completed the required educational qualifications

c)Has no criminal record

d)Meets the age requirement (21-35 years)

e)Has passed the medical fitness test

Write a program to verify job eligibility.

entrance\_exam = input("Have you cleared the entrance exam? (Yes/No): ") == "Yes"

education\_completed = input("Have you completed the required educational qualifications? (Yes/No): ") == "Yes"

criminal\_record = input("Do you have any criminal record? (Yes/No): ") == "No"

age = int(input("Enter your age: "))

medical\_test = input("Have you passed the medical fitness test? (Yes/No): ") == "Yes"

if entrance\_exam and education\_completed and criminal\_record and 21 <= age <= 35 and medical\_test:

print("Eligible for government job")

else:

print("Not eligible for government job")

Driver's License Application

A person qualifies for a driver's license if:

a)Is at least 18 years old

b)Has passed the driving test

c)Has a learner's permit

d)Has completed mandatory driving classes

e)Has no outstanding traffic fines

Write a program to check eligibility for a driver’s license.

age = int(input("Enter your age: "))

driving\_test = input("Have you passed the driving test? (Yes/No): ") == "Yes"

learner\_permit = input("Do you have a learner's permit? (Yes/No): ") == "Yes"

driving\_classes = input("Have you completed mandatory driving classes? (Yes/No): ") == "Yes"

traffic\_fines = input("Do you have any outstanding traffic fines? (Yes/No): ") == "No"

if age >= 18 and driving\_test and learner\_permit and driving\_classes and traffic\_fines:

print("Eligible for driver's license")

else:

print("Not eligible for driver's license")

Marriage Eligibility Check

A person qualifies for marriage registration if:

a)The groom is at least 21 years old

b)The bride is at least 18 years old

c)Both give mutual consent

d)Both provide valid ID proofs

e)There are no legal objections

Write a program to verify marriage eligibility.

groom\_age = int(input("Enter the groom's age: "))

bride\_age = int(input("Enter the bride's age: "))

mutual\_consent = input("Do both parties give mutual consent? (Yes/No): ") == "Yes"

id\_proof = input("Do both parties have valid ID proofs? (Yes/No): ") == "Yes"

legal\_objections = input("Are there any legal objections? (Yes/No): ") == "No"

if groom\_age >= 21 and bride\_age >= 18 and mutual\_consent and id\_proof and legal\_objections:

print("Eligible for marriage registration")

else:

print("Not eligible for marriage registration")

Loops

Password Strength Checker

Write a Python program that asks for a password and checks if it meets these conditions using loops:

a)Minimum 8 characters

b)Contains at least one uppercase letter

c)Contains at least one digit

d)Contains at least one special character (@, #, $, etc.)

e)If any condition is missing, prompt the user again.

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

while True:

password = input("Enter your password: ")

has\_min\_length = len(password) >= 8

has\_uppercase = any(char.isupper() for char in password)

has\_digit = any(char.isdigit() for char in password)

has\_special\_char = any(char in special\_characters for char in password)

if has\_min\_length and has\_uppercase and has\_digit and has\_special\_char:

print("Strong Password.")

break

else:

print("Weak Password.")

if not has\_min\_length:

print("Password must have minimum 8 characters.")

if not has\_uppercase:

print("Password must contain at least one uppercase letter.")

if not has\_digit:

print("Password must contain at least one digit.")

if not has\_special\_char:

print("Password must contain at least one special character (@, #, $, etc.).")

print("Please try again.")

Validate Employee ID Format

An employee ID should have:

a)Exactly 6 characters

b)Starts with 2 uppercase letters (e.g., EM1234)

c)Followed by 4 digits

Write a loop-based program to validate employee IDs.

while True:

employee\_id = input("Enter your employee ID: ")

has\_length = len(employee\_id) == 6

has\_start\_two\_uppercases = employee\_id[0].isupper() and employee\_id[1].isupper()

has\_last\_four\_digits = employee\_id[2:].isdigit()

if has\_length and has\_start\_two\_uppercases and has\_last\_four\_digits:

print("Valid employee ID.")

break

else:

print("Not valid employee ID.")

if not has\_length:

print("An employee ID should have exactly 6 characters.")

if not has\_start\_two\_uppercases:

print("An employee ID should start with 2 uppercase letters.")

if not has\_last\_four\_digits:

print("An employee ID should have last 4 digits.")

print("Please try again.")

Unique Product Codes Generator

Write a program that generates unique product codes using loops.

The product code should be:

a)4 uppercase letters (A-Z)

b)Followed by 4 digits (0000-9999)

c)Generate 10 unique product codes and store them in a list.

import random, string

product\_codes = []

while len(product\_codes) < 10:

letters = ''.join(random.choices(string.ascii\_uppercase, k=4))

digits = ''.join(random.choices(string.digits, k=4))

product\_code = letters + digits

if product\_code not in product\_codes:

product\_codes.append(product\_code)

print("Generated Product Codes:")

for code in product\_codes:

print(code)

Attendance Tracking System

A company has 100 employees (IDs 1 to 100). Each employee's attendance is stored in a list (1 for present, 0 for absent).

Write a loop-based program to:

a)Count total present and absent employees

b)Find the employee IDs who were absent

import random

attendance = [random.choice([0, 1]) for \_ in range(1, 101)]

present\_count = 0

absent\_count = 0

absent\_employees = []

for employee\_id, status in enumerate(attendance, start=1):

if status == 1:

present\_count += 1

else:

absent\_count += 1

absent\_employees.append(employee\_id)

print("Attendance List: ", attendance)

print("Total Present Employees: ", present\_count)

print("Total Absent Employees: ", absent\_count)

print("Absent Employee IDs: ", absent\_employees)

Detect Consecutive Failures in a System

A server logs response times (in milliseconds) for N requests. If a server gives a response time of -1, it means the request failed.

Write a program to check if the server had 3 consecutive failures and print "System Needs Attention" if found.

import random

server\_log = [random.choice([-1, random.randint(100, 500)]) for \_ in range(50)]

print("Server Log: ", server\_log)

consecutive\_failures = 0

for response\_time in server\_log:

if response\_time == -1:

consecutive\_failures += 1

if consecutive\_failures == 3:

print("System Needs Attention")

break

else:

consecutive\_failures = 0

else:

print("System is Stable")

Find First Duplicate in a List

Given a list of numbers, find the first duplicate number using a loop.

for E.g:

Input: [3, 1, 4, 2, 5, 3, 6, 7]

Output: 3

numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))

seen = set()

for num in numbers:

if num in seen:

print("First duplicate number: ", num)

break

else:

seen.add(num)

else:

print("No duplicate numbers found.")

Calculate Running Average of Sales

A company records daily sales for N days. Write a loop-based program to compute the running average after each day.

n = int(input("Enter the number of days: "))

sales = []

for \_ in range(n):

sales.append(float(input()))

running\_sum = 0

print("Running Averages:")

for day, daily\_sale in enumerate(sales, start=1):

running\_sum += daily\_sale

running\_average = running\_sum / day

print(f"Day {day}: {running\_average:.2f}")

Identify Top 3 Performers

A company has 50 employees. Their performance scores (1-100) are stored in a list.

Write a program using loops to find the top 3 performers.

import random

performance\_scores = [random.randint(1, 100) for \_ in range(50)]

print("Performance Scores: ", performance\_scores)

top\_scores = sorted(performance\_scores, reverse=True)[:3]

print("Top 3 Performers:")

for rank, score in enumerate(top\_scores, start=1):

employee\_id = performance\_scores.index(score) + 1

print(f"Rank {rank}: Employee ID {employee\_id} with Score {score}")

Detect Cyclic Pattern in a List

Write a program that checks if a list follows a repeating pattern.

Example:

Input: [1, 2, 3, 1, 2, 3, 1, 2, 3]

Output: "Pattern detected: [1, 2, 3]"

numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))

pattern\_length = 0

for i in range(1, len(numbers)):

if numbers[:i] \* (len(numbers) // i) == numbers[:len(numbers)]:

pattern\_length = i

break

if pattern\_length > 0:

print(f"Pattern detected: {numbers[:pattern\_length]}")

else:

print("No cyclic pattern detected.")

Find Missing Value in a Sequence

A sequence of numbers (1 to N) is stored in a list, but one number is missing.

Write a loop-based program to find the missing number.

Example:

Input: [1, 2, 3, 5, 6]

Output: 4

numbers = list(map(int, input("Enter sequence of numbers (1 to N) separated by spaces: ").split()))

N = len(numbers) + 1

expected\_sum = N \* (N + 1) // 2

actual\_sum = sum(numbers)

missing\_number = expected\_sum - actual\_sum

print(f"The missing number is: ", missing\_number)

Reorder Elements Based on Frequency

Given a list of numbers, sort them based on frequency of occurrence using loops.

Example:

Input: [4, 5, 6, 5, 4, 5]

Output: [5, 5, 5, 4, 4, 6] (Since 5 appears most)

from collections import Counter

numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))

frequency = Counter(numbers)

sorted\_numbers = sorted(numbers, key=lambda x: (-frequency[x], x))

print("Reordered list based on frequency: ", sorted\_numbers)

Write a pseudocode to find the sum of the series given below:

1! + 2! + 3! …. n!

import math

n = int(input("Enter the value of n: "))

sum\_of\_series = 0

for i in range(1, n + 1):

sum\_of\_series += math.factorial(i)

print("The sum of the series is:", sum\_of\_series)

Write a pseudocode to input the value for x and n and print the following series:

1 + x + x

2 + …. + x

n

x = int(input("Enter the value of x: "))

n = int(input("Enter the value of n: "))

sum\_of\_series = 0

print("Series: ", end="")

for i in range(0, n + 1):

term = x \*\* i

print(term, end=" ")

sum\_of\_series += term

print("The sum of the series is:", sum\_of\_series)

Display the sum and average of all odd numbers divisible by 3 that fall within the range 245 to 999, both inclusive.

sum\_of\_numbers = 0

count = 0

for i in range(245, 1000):

if i % 2 != 0 and i % 3 == 0:

sum\_of\_numbers += i

count += 1

average = sum\_of\_numbers / count if count != 0 else 0

print("The sum is:", sum\_of\_numbers)

print("The average is:", average)

Accept a 5 digit decimal number as input and display the number in reverse order.

(Example: if input is 12345, then output must be 54321)

num = int(input("Enter a 5-digit number: "))

reverse = 0

while num > 0:

digit = num % 10

reverse = reverse \* 10 + digit

num = num // 10

print("The reversed number is:", reverse)

Find Employee ID with the Most Logins

A list contains multiple employee login records in a day (e.g., ["E123", "E456", "E123", "E789", "E123"]).

a)Write a program using loops to count the login frequency.

b)Display the employee ID with the most logins.

logins = input("Enter the login records separated by spaces (e.g., 'E123 E456 E123'): ").split()

login\_count = {}

for id in logins:

login\_count[id] = login\_count.get(id, 0) + 1

most\_logins = max(login\_count, key=login\_count.get)

print("Login frequency:", login\_count)

print("Employee ID with the most logins:", most\_logins)

Simulate a Parking System

A parking lot has 10 slots (numbered 1-10).

a)If a car enters, assign the first available slot.

b)If a car exits, free up the slot.

c)If full, print "Parking Full".

Use loops to handle multiple entries/exits.

parking\_slots = [0] \* 10

while True:

print("Current parking status:", parking\_slots)

action = input("Enter 'enter' for car entry, 'exit' for car exit, or 'stop' to end: ").strip().lower()

if action == "enter":

assigned = False

for i in range(10):

if parking\_slots[i] == 0:

parking\_slots[i] = 1

print(f"Car parked in slot {i + 1}")

assigned = True

break

if not assigned:

print("Parking Full")

elif action == "exit":

slot = int(input("Enter the slot number to free up (1-10): "))

if 1 <= slot <= 10:

if parking\_slots[slot - 1] == 1:

parking\_slots[slot - 1] = 0

print(f"Slot {slot} is now free")

else:

print(f"Slot {slot} is already empty")

else:

print("Invalid slot number. Please enter a number between 1 and 10.")

elif action == "stop":

print("Exiting the parking system. Goodbye!")

break

else:

print("Invalid input. Please try again.")

Find the Most Frequent Pair of Consecutive Elements

Given a sequence, find the most frequently occurring consecutive pair using

Example:

Input: [1, 2, 2, 3, 2, 2, 3, 4]

Output: "Most common pair: (2, 2)"

sequence = list(map(int, input("Enter the sequence of numbers separated by spaces: ").split()))

pair\_count = {}

for i in range(len(sequence) - 1):

pair = (sequence[i], sequence[i + 1])

pair\_count[pair] = pair\_count.get(pair, 0) + 1

most\_common\_pair = max(pair\_count, key=pair\_count.get)

print("Most common pair:", most\_common\_pair)

Print Spiral Pattern of Numbers

Write a loop-based program to generate a spiral matrix.

Example for N = 3:

1 2 3

8 9 4

7 6 5

n = int(input("Enter the size of the spiral matrix (e.g., 3 for a 3x3 matrix): "))

matrix = [[0] \* n for \_ in range(n)]

num = 1

top, left = 0, 0

bottom, right = n - 1, n - 1

while num <= n \* n:

# Fill the top row

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

matrix[top][i] = num

num += 1

top += 1

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

matrix[i][right] = num

num += 1

right -= 1

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

matrix[bottom][i] = num

num += 1

bottom -= 1

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

matrix[i][left] = num

num += 1

left += 1

for row in matrix:

print(" ".join(map(str, row)))

Find the difference between two valid dates. (output should be in days)

from datetime import datetime

date1 = input("Enter the first date (YYYY-MM-DD): ")

date2 = input("Enter the second date (YYYY-MM-DD): ")

date1 = datetime.strptime(date1, "%Y-%m-%d")

date2 = datetime.strptime(date2, "%Y-%m-%d")

difference = abs((date2 - date1).days)

print("The difference between the two dates is:", difference, "days")

String Validation

Reverse Words in a Sentence

Write a function to reverse the order of words in a sentence without using built-in .split() and .join().

Example:

Input: "Welcome to Python"

Output: "Python to Welcome"

sentence = input("Enter a sentence: ")

words = []

word = ""

for char in sentence:

if char == " ":

words.append(word)

word = ""

else:

word += char

words.append(word)

reversed\_sentence = ""

for i in range(len(words) - 1, -1, -1):

reversed\_sentence += words[i] + (" " if i != 0 else "")

print("Reversed sentence:", reversed\_sentence)

Capitalize Each Word in a Sentence

Write a program to capitalize the first letter of every word in a sentence (without using .title()).

Example:

Input: "python programming is fun"

Output: "Python Programming Is Fun"

sentence = input("Enter a sentence: ")

capitalized\_sentence = ""

capitalize = True

for char in sentence:

if capitalize and char.isalpha():

capitalized\_sentence += char.upper()

capitalize = False

else:

capitalized\_sentence += char

if char == " ":

capitalize = True

print("Capitalized sentence:", capitalized\_sentence)

Find the Most Frequent Character in a String

Given a string, find the most frequently occurring character.

Example:Input: "success"

Output: "Most frequent character: s"

string = input("Enter a string: ")

char\_count = {}

for char in string:

char\_count[char] = char\_count.get(char, 0) + 1

most\_frequent = max(char\_count, key=char\_count.get)

print("Most frequent character:", most\_frequent)

Check if a String is a Valid Email

Validate whether a given string is a proper email address.

Conditions:

a)Should contain exactly one "@"

b)Should end with a valid domain (.com, .in, .org)

Example:

Input: "john.doe@email.com"

Output: "Valid email"

email = input("Enter an email address: ")

if email.count("@") == 1 and email.endswith((".com", ".in", ".org")):

print("Valid email")

else:

print("Invalid email")

Validate a Strong Password

A password is considered strong if it:

a)Has at least 8 characters

b)Contains at least one uppercase letter

c)Contains at least one lowercase letter

d)Contains at least one digit

e)Contains at least one special character (@, #, $, etc.)

Write a program to validate passwords based on these conditions.

import re

password = input("Enter a password: ")

if len(password) >= 8 and re.search(r'[A-Z]', password) and re.search(r'[a-z]', password) and re.search(r'\d', password) and re.search(r'[@#$]', password):

print("Strong password")

else:

print("Weak password")

Check if a String is a Valid URL

Write a function to check if a given string is a valid URL.

Valid examples:

https://www.google.com

http://example.org

www.github.com

url = input("Enter a URL: ")

if url.startswith(("http://", "https://", "www.")):

print("Valid URL")

else:

print("Invalid URL")

Validate a PAN Card Number (India)

A valid PAN card number has the format AAAAA9999A, where:

a)First 5 characters are uppercase letters

b)Next 4 characters are digits

c)Last character is an uppercase letter

Example:

Input: "ABCDE1234F"

Output: "Valid PAN number"

pan\_number = input("Enter a PAN card number: ")

if len(pan\_number) == 10 and pan\_number[:5].isupper() and pan\_number[5:9].isdigit() and pan\_number[9].isupper():

print("Valid PAN number")

else:

print("Invalid PAN number")

Find Longest Word in a Sentence

Write a function to find the longest word in a sentence.

sentence = input("Enter a sentence: ")

longest\_word = ""

word = ""

for char in sentence + " ":

if char == " ":

if len(word) > len(longest\_word):

longest\_word = word

word = ""

else:

word += char

print("Longest word:", longest\_word)

Count the Number of Words in a String Without Using split()

Write a program to count the number of words in a sentence without using .split().

Example:

Input: "Python is awesome!"

Output: 3

sentence = input("Enter a sentence: ")

count = 0

in\_word = False

for char in sentence:

if char == " ":

in\_word = False

elif not in\_word:

count += 1

in\_word = True

print("Number of words:", count)

Replace All Occurrences of a Word in a Sentence

Write a program to replace all occurrences of a word with another word.

Example:

Input: "The sky is blue. The ocean is blue."

Replace "blue" with "clear"

Output: "The sky is clear. The ocean is clear."

sentence = input("Enter a sentence: ")

old\_word = input("Enter the word to replace: ")

new\_word = input("Enter the new word: ")

result = ""

word = ""

for char in sentence + " ":

if char == " ":

if word == old\_word:

result += new\_word + " "

else:

result += word + " "

word = ""

else:

word += char

print("Updated sentence:", result.strip())

Read a string. Store all the characters in even position in one array and odd positions

in another array.

string = input("Enter a string: ")

even\_positions = []

odd\_positions = []

for i in range(len(string)):

if i % 2 == 0:

even\_positions.append(string[i])

else:

odd\_positions.append(string[i])

print("Even position characters:", even\_positions)

print("Odd position characters:", odd\_positions)

A valid Indian vehicle registration number follows this format:

a)Two uppercase letters (State Code)

b)Two digits (RTO Code)

c)Two uppercase letters (Series Code)

d)Four digits (Vehicle Number)

Example:

Input: "TN10AB1234"

Output: "Valid vehicle number"

vehicle\_number = input("Enter a vehicle registration number: ")

if len(vehicle\_number) == 10 and vehicle\_number[:2].isupper() and vehicle\_number[2:4].isdigit() and vehicle\_number[4:6].isupper() and vehicle\_number[6:].isdigit():

print("Valid vehicle number")

else:

print("Invalid vehicle number")

Validate a Credit Card Number

A valid credit card number should:

a)Be exactly 16 digits

b)Start with 4 (Visa), 5 (MasterCard), or 6 (RuPay)

Input: "4123456789012345"

Output: "Valid Credit Card"

card\_number = input("Enter a credit card number: ")

if len(card\_number) == 16 and card\_number[0] in "456" and card\_number.isdigit():

print("Valid Credit Card")

else:

print("Invalid Credit Card")

Check if a String Contains Only Alphanumeric Characters

Write a program to validate if a string contains only alphabets and numbers (A-Z, a-z, 0-9).

Example:

Input: "Python3Rocks"

Output: "Valid"

string = input("Enter a string: ")

if string.isalnum():

print("Valid")

else:

print("Invalid")

Validate a Date Format (DD-MM-YYYY)

Write a function to check if a given date string follows the format DD-MM-YYYY.

Example:

Input: "25-12-2024"

Output: "Valid Date" "

date = input("Enter a date (DD-MM-YYYY): ")

if len(date) == 10 and date[2] == "-" and date[5] == "-" and date[:2].isdigit() and date[3:5].isdigit() and date[6:].isdigit():

print("Valid Date")

else:

print("Invalid Date")

Functions

Identify Function prototype for the following requirements.

a) Online shopping

b) E-ticket application

c) Fund transfer application

d) Mobile Recharge application

def online\_shopping(cart\_items: list, payment\_method: str) -> str:

print(f"Items in your cart: {cart\_items}")

print(f"Payment method chosen: {payment\_method}")

return "Purchase successful!"

cart\_items = input("Enter items in your cart, separated by commas: ").split(",")

payment\_method = input("Enter payment method (e.g., Credit Card, PayPal): ")

output = online\_shopping(cart\_items, payment\_method)

print(output)

def e\_ticket\_application(user\_details: dict, event\_id: str) -> str:

print(f"User Details: {user\_details}")

print(f"Event ID: {event\_id}")

return "E-ticket issued successfully!"

user\_name = input("Enter your name: ")

user\_email = input("Enter your email: ")

event\_id = input("Enter the event ID: ")

user\_details = {"Name": user\_name, "Email": user\_email}

output = e\_ticket\_application(user\_details, event\_id)

print(output)

def fund\_transfer(sender\_account: str, receiver\_account: str, amount: float) -> bool:

print(f"Sender Account: {sender\_account}")

print(f"Receiver Account: {receiver\_account}")

print(f"Amount: {amount}")

return True

sender\_account = input("Enter sender account number: ")

receiver\_account = input("Enter receiver account number: ")

amount = float(input("Enter transfer amount: "))

success = fund\_transfer(sender\_account, receiver\_account, amount)

if success:

print("Fund transfer successful!")

else:

print("Fund transfer failed.")

def mobile\_recharge(mobile\_number: str, plan\_id: str) -> str:

print(f"Mobile Number: {mobile\_number}")

print(f"Plan ID: {plan\_id}")

return "Recharge successful!"

mobile\_number = input("Enter your mobile number: ")

plan\_id = input("Enter recharge plan ID: ")

output = mobile\_recharge(mobile\_number, plan\_id)

print(output)

Create a function that takes a sentence as parameter and return the length of the second word in the sentence.

def second\_word\_length(sentence: str) -> int:

word\_count = 0

word = ""

for char in sentence + " ":

if char == " ":

word\_count += 1

if word\_count == 2:

return len(word)

word = ""

else:

word += char

return 0

sentence = input("Enter a sentence: ")

print("Length of the second word:", second\_word\_length(sentence))

Basic Class and Object Concepts

Create a class Car with attributes brand, model, and year. Instantiate an object of this class and print its details.

class Car:

def \_\_init\_\_(self, brand, model, year):

self.brand = brand

self.model = model

self.year = year

def \_\_str\_\_(self):

return f"Car: {self.year} {self.brand} {self.model}"

brand = input("Enter car brand: ")

model = input("Enter car model: ")

year = int(input("Enter car year: "))

car = Car(brand, model, year)

print(car)

Write a class Rectangle that takes length and breadth as parameters. Create a method to calculate the area of the rectangle. Instantiate two rectangles and compare their areas.

class Rectangle:

def \_\_init\_\_(self, length, breadth):

self.length = length

self.breadth = breadth

def area(self):

return self.length \* self.breadth

print("Enter dimensions for Rectangle 1:")

length1 = float(input("Length: "))

breadth1 = float(input("Breadth: "))

b = Rectangle(length1, breadth1)

print("Enter dimensions for Rectangle 2:")

length2 = float(input("Length: "))

breadth2 = float(input("Breadth: "))

c = Rectangle(length2, breadth2)

if b.area() > c.area():

print("Rectangle 1 has larger area")

elif b.area() < c.area():

print("Rectangle 2 has larger area")

else:

print("Both rectangles have equal area")

Create a class Student with attributes name, roll\_number, and marks. Define a method to display the student's details. Instantiate the class and print the details of a student.

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name

self.roll\_number = roll\_number

self.marks = marks

def display\_details(self):

print(f"Student Name: {self.name}")

print(f"Roll Number: {self.roll\_number}")

print(f"Marks: {self.marks}")

name = input("Enter student name: ")

roll\_number = input("Enter roll number: ")

marks = float(input("Enter marks: "))

student = Student(name, roll\_number, marks)

student.display\_details()

Encapsulation (Data Hiding & Getters/Setters)

Create a class BankAccount with a private attribute \_balance. Implement methods to deposit and withdraw money while ensuring the balance never goes negative.

class BankAccount:

def \_\_init\_\_(self, initial\_balance=0):

self.\_balance = initial\_balance

def deposit(self, amount):

if amount > 0:

self.\_balance += amount

return True

return False

def withdraw(self, amount):

if amount > 0 and self.\_balance >= amount:

self.\_balance -= amount

return True

return False

def get\_balance(self):

return self.\_balance

initial = float(input("Enter initial balance: "))

account = BankAccount(initial)

deposit = float(input("Enter amount to deposit: "))

account.deposit(deposit)

withdraw = float(input("Enter amount to withdraw: "))

account.withdraw(withdraw)

print("Current Balance:", account.get\_balance())

Modify the BankAccount class to include a getter method for checking the balance and a setter method to update it (ensuring balance cannot be set to a negative value).

class BankAccount:

def \_\_init\_\_(self, initial\_balance=0):

self.\_balance = initial\_balance

def get\_balance(self):

return self.\_balance

def set\_balance(self, value):

if value >= 0:

self.\_balance = value

else:

print("Balance cannot be negative")

initial = float(input("Enter initial balance: "))

account = BankAccount(initial)

print("Current Balance:", account.get\_balance())

new\_balance = float(input("Enter new balance: "))

account.set\_balance(new\_balance)

print("Updated Balance:", account.get\_balance())

Create a class Laptop with private attributes brand and price. Provide getter and setter methods to access and modify these attributes.

class Laptop:

def \_\_init\_\_(self, brand, price):

self.\_brand = brand

self.\_price = price

@property

def brand(self):

return self.\_brand

@brand.setter

def brand(self, value):

self.\_brand = value

@property

def price(self):

return self.\_price

@price.setter

def price(self, value):

if value > 0:

self.\_price = value

else:

print("Price must be positive")

brand = input("Enter laptop brand: ")

price = float(input("Enter laptop price: "))

laptop = Laptop(brand, price)

print("Laptop Details:", laptop.brand, laptop.price)

new\_price = float(input("Enter new price: "))

laptop.price = new\_price

Inheritance (Code Reusability & Hierarchy)

Create a class Animal with a method make\_sound(). Derive subclasses Dog and Cat from it, and override make\_sound() to print respective sounds.

class Animal:

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

self.name = name

def make\_sound(self):

return f"{self.name} makes a sound"

class Dog(Animal):

def make\_sound(self):

return f"{self.name} barks"

class Cat(Animal):

def make\_sound(self):

return f"{self.name} meows"

name = input("Enter the animal's name: ")

dog = Dog(name)

print(dog.make\_sound ())

cat = Cat(name)

print(cat.make\_sound ())

Create a base class Employee with attributes name and salary. Derive two classes: Manager and Developer, where Manager has an additional attribute team\_size and Developer has programming\_language.

class Employee:

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

class Manager(Employee):

def \_\_init\_\_(self, name, salary, team\_size):

super().\_\_init\_\_(name, salary)

self.team\_size = team\_size

class Developer(Employee):

def \_\_init\_\_(self, name, salary, programming\_language):

super().\_\_init\_\_(name, salary)

self.programming\_language = programming\_language

m\_name = input("Enter Manager's name: ")

m\_salary = float(input("Enter Manager's salary: "))

m\_team\_size = int(input("Enter Manager's team size: "))

manager = Manager(m\_name, m\_salary, m\_team\_size)

d\_name = input("Enter Developer's name: ")

d\_salary = float(input("Enter Developer's salary: "))

d\_language = input("Enter Developer's programming language: ")

developer = Developer(d\_name, d\_salary, d\_language)

print("Manager Details:")

print(f"Name: {manager.name}, Salary: {manager.salary}, Team Size: {manager.team\_size}")

print("Developer Details:")

print(f"Name: {developer.name}, Salary: {developer.salary}, Language: {developer.programming\_language}")

Create a class Shape with a method area(). Inherit Circle and Square classes from Shape and implement their respective area calculations.

class Shape:

def area(self):

pass

class Circle(Shape):

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

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

class Square(Shape):

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

self.side = side

def area(self):

return self.side \*\* 2

radius = float(input("Enter the radius of the circle: "))

side = float(input("Enter the side length of the square: "))

circle = Circle(radius)

square = Square(side)

print(f"Area of Circle: {circle.area():.2f}")

print(f"Area of Square: {square.area():.2f}")

Create a class Vehicle with attributes brand and speed. Derive Car and Bike classes from it and add specific attributes and methods to each.

class Vehicle:

def \_\_init\_\_(self, brand, speed):

self.brand = brand

self.speed = speed

def display(self):

print(f"Brand: {self.brand}, Speed: {self.speed} km/h")

class Car(Vehicle):

def \_\_init\_\_(self, brand, speed, num\_doors):

super().\_\_init\_\_(brand, speed)

self.num\_doors = num\_doors

def display(self):

super().display()

print(f"Number of Doors: {self.num\_doors}")

class Bike(Vehicle):

def \_\_init\_\_(self, brand, speed, has\_basket):

super().\_\_init\_\_(brand, speed)

self.has\_basket = has\_basket

def display(self):

super().display()

print(f"Has Basket: {'Yes' if self.has\_basket else 'No'}")

car\_brand = input("Enter car brand: ")

car\_speed = int(input("Enter car speed (in km/h): "))

car\_doors = int(input("Enter number of doors: "))

car = Car(car\_brand, car\_speed, car\_doors)

bike\_brand = input("Enter bike brand: ")

bike\_speed = int(input("Enter bike speed (in km/h): "))

bike\_basket\_input = input("Does the bike have a basket? (yes/no): ").strip().lower()

bike\_basket = bike\_basket\_input == "yes"

bike = Bike(bike\_brand, bike\_speed, bike\_basket)

print(“Vehicle Information”)

car.display()

print()

bike.display()

Polymorphism (Method Overriding & Operator Overloading)

Modify the Shape class hierarchy to include a display() method that prints shape details. Override this method in Circle and Square.

Directory Name: Q13

Package Name: baseentity

File Name: shape.py

class Shape:

def area(self):

pass

def display(self):

print("This is a generic shape")

Package Name: derivedentities

File Name: circle.py

from Q13.baseentity.shape import Shape

class Circle(Shape):

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

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

def display(self):

print(f"This is a circle with radius {self.radius}")

File Name: square.py

from Q13.baseentity.shape import Shape

class Square(Shape):

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

self.side = side

def area(self):

return self.side \*\* 2

def display(self):

print(f"This is a square with side {self.side}")

Package Name: main

File Name: main.py

from Q13.derivedentities.circle import Circle

from Q13.derivedentities.square import Square

if \_\_name\_\_ == "\_\_main\_\_":

shapes = []

num\_shapes = int(input("How many shapes do you want to create?: "))

for \_ in range(num\_shapes):

shape\_type = input("Enter the shape type (Circle/Square): ").strip().lower()

if shape\_type == "circle":

radius = float(input("Enter radius: "))

shapes.append(Circle(radius))

elif shape\_type == "square":

side = float(input("Enter side: "))

shapes.append(Square(side))

else:

print("Invalid shape type.")

for shape in shapes:

shape.display()

print("Area:", shape.area())

Create a function calculate\_total\_price() that takes different objects (Laptop, Mobile, Tablet) and calls their method get\_price() to calculate the total price dynamically.

Directory Name: Q14

Package Name: baseentity

File Name: calculator.py

def calculate\_total\_price(products):

return sum(product.get\_price() for product in products)

Package Name: derivedentities

File Name: laptop.py

class Laptop:

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

self.price = price

def get\_price(self):

return self.price

File Name: mobile.py

class Mobile:

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

self.price = price

def get\_price(self):

return self.price

File Name: tablet.py

class Tablet:

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

self.price = price

def get\_price(self):

return self.price

Package Name: main

File Name: main.py

from Q14.derivedentities.laptop import Laptop

from Q14.derivedentities.mobile import Mobile

from Q14.derivedentities.tablet import Tablet

from Q14.baseentity.calculator import calculate\_total\_price

if \_\_name\_\_ == '\_\_main\_\_':

items = []

n = int(input("How many products? "))

for \_ in range(n):

p\_type = input("Enter product type (laptop/mobile/tablet): ").strip().lower()

price = float(input("Enter price: "))

if p\_type == "laptop":

items.append(Laptop(price))

elif p\_type == "mobile":

items.append(Mobile(price))

elif p\_type == "tablet":

items.append(Tablet(price))

else:

print("The product type is unknown.")

print("Total price:", calculate\_total\_price(items))

Create a class Person with a method introduce(). Override it in subclasses Teacher and Student to print different introductions.

Directory Name: Q15

Package Name: baseentity

File Name: person.py

class Person:

def introduce(self):

print("I am a person")

Package Name: derivedentities

File Name: student.py

from Q15.baseentity.person import Person

class Student(Person):

def introduce(self):

print("I am a student")

File Name: teacher.py

from Q15.baseentity.person import Person

class Teacher(Person):

def introduce(self):

print("I am a teacher")

Package Name: main

File Name: main.py

from Q15.baseentity.person import Person

from Q15.derivedentities.student import Student

from Q15.derivedentities.teacher import Teacher

if \_\_name\_\_ == '\_\_main\_\_':

people = []

num\_people = int(input("How many people to introduce?: "))

for \_ in range(num\_people):

role = input("Enter the role (Person/Teacher/Student): ").strip().lower()

if role == "person":

people.append(Person())

elif role == "teacher":

people.append(Teacher())

elif role == "student":

people.append(Student())

else:

print("The role is invalid.")

print("Introductions")

for person in people:

person.introduce()

Overload the \_\_str\_\_ method in a class Bank to return a formatted string when the object is printed.

class Bank:

def \_\_init\_\_(self, name, total\_customers):

self.name = name

self.total\_customers = total\_customers

def \_\_str\_\_(self):

return f"Bank: {self.name} (Customers: {self.total\_customers})"

bank\_name = input("Enter bank name: ")

customers = int(input("Enter total number of customers: "))

bank = Bank(bank\_name, customers)

print("Bank Info")

print(bank)

Abstraction (Hiding Implementation Details using ABC)

Create an abstract class Appliance with an abstract method turn\_on(). Implement subclasses WashingMachine and Refrigerator with specific functionality.

Directory Name: Q17

Package Name: baseentity

File Name: appliance.py

from abc import ABC, abstractmethod

class Appliance(ABC):

@abstractmethod

def turn\_on(self):

pass

Package Name: derivedentities

File Name: refrigerator.py

from Q17.baseentity.appliance import Appliance

class Refrigerator(Appliance):

def turn\_on(self):

print("Refrigerator started. Cooling initiated...")

File Name: washingmachine.py

from Q17.baseentity.appliance import Appliance

class WashingMachine(Appliance):

def turn\_on(self):

print("Washing machine started. Water filling...")

Package Name: main

File Name: main.py

from Q17.derivedentities.refrigerator import Refrigerator

from Q17.derivedentities.washingmachine import WashingMachine

if \_\_name\_\_ == '\_\_main\_\_':

choice = input("Enter appliance to turn on (washing/refrigerator): ").strip().lower()

if choice == "washing":

appliance = WashingMachine()

elif choice == "refrigerator":

appliance = Refrigerator()

else:

print("The choice is invalid.")

appliance = None

if appliance:

appliance.turn\_on()

Create an abstract class Animal with an abstract method move(). Implement Bird and Fish subclasses with their movement styles.

from abc import ABC, abstractmethod

class Animal(ABC):

@abstractmethod

def move(self):

pass

class Bird(Animal):

def move(self):

print("Bird is flying")

class Fish(Animal):

def move(self):

print("Fish is swimming")

choice = input("Enter animal to simulate movement (bird/fish): ").strip().lower()

if choice == "bird":

animal = Bird()

elif choice == "fish":

animal = Fish()

else:

print("Invalid choice.")

animal = None

if animal:

animal.move()

Create an abstract class Payment with an abstract method process\_payment(). Implement CreditCardPayment and PayPalPayment subclasses.

Directory Name: Q19

Package Name: baseentity

File Name: payment.py

from abc import ABC, abstractmethod

class Payment(ABC):

@abstractmethod

def process\_payment(self, amount):

pass

Package Name: derivedentities

File Name: creditcard.py

from Q19.baseentity.payment import Payment

class CreditCardPayment(Payment):

def process\_payment(self, amount):

print(f"Processing credit card payment of Rs {amount}")

File Name: paypal.py

from Q19.baseentity.payment import Payment

class PayPalPayment(Payment):

def process\_payment(self, amount):

print(f"Processing PayPal payment of Rs {amount}")

Package Name: main

File Name: main.py

from Q19.derivedentities.paypal import PayPalPayment

from Q19.derivedentities.creditcard import CreditCardPayment

if \_\_name\_\_ == "\_\_main\_\_":

method = input("Choose payment method (credit/paypal): ").strip().lower()

amount = float(input("Enter amount to pay: "))

if method == "credit":

payment = CreditCardPayment()

elif method == "paypal":

payment = PayPalPayment()

else:

print("Invalid method.")

payment = None

if payment:

payment.process\_payment(amount)

Shop has products, implement the requirement using collections

Directory Name: List

Sub-directory Name: Q20

Package Name: ProductDTO

File Name: product.py

class Product(object):

def \_\_init\_\_(self, product\_id, name, price):

self.\_\_product\_id = product\_id

self.\_\_name = name

self.\_\_price = price

def get\_product\_id(self):

return self.\_\_product\_id

def get\_name(self):

return self.\_\_name

def get\_price(self):

return self.\_\_price

def set\_name(self, name):

self.\_\_name = name

def set\_price(self, price):

self.\_\_price = price

def del\_product\_id(self):

del self.\_\_product\_id

def del\_name(self):

del self.\_\_name

def del\_price(self):

del self.\_\_price

def \_\_str\_\_(self):

return f"Product ID: {self.\_\_product\_id} | Name: {self.\_\_name} | Price: Rs.{self.\_\_price}"

Package Name: ShopDTO

File Name: shop.py

from List.Q20.ShopException.shopexception import ProductNotFoundError

class Shop(object):

def \_\_init\_\_(self, shop\_id, shop\_name):

self.\_\_shop\_id = shop\_id

self.\_\_shop\_name = shop\_name

self.\_\_product\_list = []

def \_\_str\_\_(self):

result = f"Shop ID: {self.\_\_shop\_id} | Shop Name: {self.\_\_shop\_name}\nProducts:\n"

result += "\n".join(str(product) for product in self.\_\_product\_list)

return result

def get\_shop\_id(self):

return self.\_\_shop\_id

def get\_shop\_name(self):

return self.\_\_shop\_name

def get\_product\_list(self):

return self.\_\_product\_list

def set\_shop\_id(self, value):

self.\_\_shop\_id = value

def set\_shop\_name(self, value):

self.\_\_shop\_name = value

def set\_product\_list(self, value):

self.\_\_product\_list = value

def add\_product\_toList(self, product\_obj):

self.\_\_product\_list.append(product\_obj)

def read\_product\_fromList(self, product\_id):

for product in self.\_\_product\_list:

if product.get\_product\_id() == product\_id:

return product

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

def update\_product\_toList(self, product\_id, new\_name, new\_price):

product\_obj = self.read\_product\_fromList(product\_id)

product\_obj.set\_name(new\_name)

product\_obj.set\_price(new\_price)

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

product = self.read\_product\_fromList(product\_id)

self.\_\_product\_list.remove(product)

Package Name: ShopException

File Name: shopexception.py

class ProductNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Product not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: ShopMain

File Name: MainModule.py

import logging

from List.Q20.ProductDTO.product import Product

from List.Q20.ShopDTO.shop import Shop

from List.Q20.ShopException.shopexception import ProductNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="shop\_inventory.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

shop1 = Shop("S01", "Croma")

shop2 = Shop("S02", "HomeCentre")

s1\_p1 = Product("P101", "Laptop", 70000)

s1\_p2 = Product("P102", "Smartphone", 25000)

s1\_p3 = Product("P103", "Tablet", 15000)

s2\_p1 = Product("P201", "Mixer", 3500)

s2\_p2 = Product("P202", "Cooker", 2200)

s2\_p3 = Product("P203", "Refrigerator", 42000)

for product in [s1\_p1, s1\_p2, s1\_p3]:

shop1.add\_product\_toList(product)

for product in [s2\_p1, s2\_p2, s2\_p3]:

shop2.add\_product\_toList(product)

shops = [shop1, shop2]

for shop in shops:

logging.info(f"Shop ID: {shop.get\_shop\_id()} | Shop Name: {shop.get\_shop\_name()}")

for product in shop.get\_product\_list():

logging.info(product)

try:

shop1.update\_product\_toList("P103", "Android Tablet", 17000)

logging.info("Shop1: Product P103 is updated.")

except ProductNotFoundError as e:

logging.error(e)

try:

shop1.remove\_product\_fromList("P102")

logging.warning("Shop1: Product P102 is removed.")

except ProductNotFoundError as e:

logging.error(e)

logging.info(f"Updated list for Shop ID: {shop1.get\_shop\_id()} | Shop Name: {shop1.get\_shop\_name()}")

for product in shop1.get\_product\_list():

logging.info(product)

Directory Name: Dictionary

Sub-directory Name: Q20

Package Name: ProductDTO

File Name: product.py

class Product(object):

def \_\_init\_\_(self, product\_id, name, price):

self.\_\_product\_id = product\_id

self.\_\_name = name

self.\_\_price = price

def get\_product\_id(self):

return self.\_\_product\_id

def get\_name(self):

return self.\_\_name

def get\_price(self):

return self.\_\_price

def set\_name(self, name):

self.\_\_name = name

def set\_price(self, price):

self.\_\_price = price

def del\_product\_id(self):

del self.\_\_product\_id

def del\_name(self):

del self.\_\_name

def del\_price(self):

del self.\_\_price

def \_\_str\_\_(self):

return f"Product ID: {self.\_\_product\_id} | Name: {self.\_\_name} | Price: Rs.{self.\_\_price}"

Package Name: ShopDTO

File Name: shop.py

from Dictionary.Q20.ShopException.shopexception import ProductNotFoundError

class Shop(object):

def \_\_init\_\_(self, shop\_id, shop\_name):

self.\_\_shop\_id = shop\_id

self.\_\_shop\_name = shop\_name

self.\_\_product\_dict = {}

def \_\_str\_\_(self):

result = f"Shop ID: {self.\_\_shop\_id} | Shop Name: {self.\_\_shop\_name}\nProducts:\n"

result += "\n".join(str(product) for product in self.\_\_product\_dict.values())

return result

def get\_shop\_id(self):

return self.\_\_shop\_id

def get\_shop\_name(self):

return self.\_\_shop\_name

def get\_product\_list(self): # Still return list for iteration

return list(self.\_\_product\_dict.values())

def set\_shop\_id(self, value):

self.\_\_shop\_id = value

def set\_shop\_name(self, value):

self.\_\_shop\_name = value

def set\_product\_dict(self, value): # If needed to bulk update

self.\_\_product\_dict = {p.get\_product\_id(): p for p in value}

def add\_product\_toDict(self, product\_obj):

self.\_\_product\_dict[product\_obj.get\_product\_id()] = product\_obj

def read\_product\_fromDict(self, product\_id):

if product\_id in self.\_\_product\_dict:

return self.\_\_product\_dict[product\_id]

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

def update\_product\_toDict(self, product\_id, new\_name, new\_price):

product\_obj = self.read\_product\_fromDict(product\_id)

product\_obj.set\_name(new\_name)

product\_obj.set\_price(new\_price)

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

if product\_id in self.\_\_product\_dict:

del self.\_\_product\_dict[product\_id]

else:

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

Package Name: ShopException

File Name: shopexception.py

class ProductNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Product not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: ShopMain

File Name: MainModule.py

import logging

from Dictionary.Q20.ProductDTO.product import Product

from Dictionary.Q20.ShopDTO.shop import Shop

from Dictionary.Q20.ShopException.shopexception import ProductNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="shop\_inventory.log", level=logging.DEBUG, format="%(asctime)s - %(levelname)s - %(message)s")

shop1 = Shop("S01", "Croma")

shop2 = Shop("S02", "HomeCentre")

s1\_p1 = Product("P101", "Laptop", 70000)

s1\_p2 = Product("P102", "Smartphone", 25000)

s1\_p3 = Product("P103", "Tablet", 15000)

s2\_p1 = Product("P201", "Mixer", 3500)

s2\_p2 = Product("P202", "Cooker", 2200)

s2\_p3 = Product("P203", "Refrigerator", 42000)

for product in [s1\_p1, s1\_p2, s1\_p3]:

shop1.add\_product\_toDict(product)

for product in [s2\_p1, s2\_p2, s2\_p3]:

shop2.add\_product\_toDict(product)

shops = [shop1, shop2]

for shop in shops:

logging.info(f"Shop ID: {shop.get\_shop\_id()} | Shop Name: {shop.get\_shop\_name()}")

for product in shop.get\_product\_list():

logging.info(product)

try:

shop1.update\_product\_toDict("P103", "Android Tablet", 17000)

logging.info("Shop1: Product P103 is updated.")

except ProductNotFoundError as e:

logging.error(e)

try:

shop1.remove\_product\_fromDict("P102")

logging.warning("Shop1: Product P102 is removed.")

except ProductNotFoundError as e:

logging.error(e)

logging.info(f"Updated list for Shop ID: {shop1.get\_shop\_id()} | Shop Name: {shop1.get\_shop\_name()}")

for product in shop1.get\_product\_list():

logging.info(product)

Directory Name: Set

Sub-directory Name: Q20

Package Name: ProductDTO

File Name: product.py

class Product(object):

def \_\_init\_\_(self, product\_id, name, price):

self.\_\_product\_id = product\_id

self.\_\_name = name

self.\_\_price = price

def get\_product\_id(self):

return self.\_\_product\_id

def get\_name(self):

return self.\_\_name

def get\_price(self):

return self.\_\_price

def set\_name(self, name):

self.\_\_name = name

def set\_price(self, price):

self.\_\_price = price

def del\_product\_id(self):

del self.\_\_product\_id

def del\_name(self):

del self.\_\_name

def del\_price(self):

del self.\_\_price

def \_\_str\_\_(self):

return f"Product ID: {self.\_\_product\_id} | Name: {self.\_\_name} | Price: Rs.{self.\_\_price}"

Package Name: ShopDTO

File Name: shop.py

from Set.Q20.ShopException.shopexception import ProductNotFoundError

class Shop(object):

def \_\_init\_\_(self, shop\_id, shop\_name):

self.\_\_shop\_id = shop\_id

self.\_\_shop\_name = shop\_name

self.\_\_product\_list = []

self.\_\_product\_id\_set = set()

def \_\_str\_\_(self):

result = f"Shop ID: {self.\_\_shop\_id} | Shop Name: {self.\_\_shop\_name}\nProducts:\n"

result += "\n".join(str(product) for product in self.\_\_product\_list)

return result

def get\_shop\_id(self):

return self.\_\_shop\_id

def get\_shop\_name(self):

return self.\_\_shop\_name

def get\_product\_list(self):

return self.\_\_product\_list

def set\_shop\_id(self, value):

self.\_\_shop\_id = value

def set\_shop\_name(self, value):

self.\_\_shop\_name = value

def set\_product\_list(self, value):

self.\_\_product\_list = value

self.\_\_product\_id\_set = {p.get\_product\_id() for p in value}

def add\_product\_toSet(self, product\_obj):

product\_id = product\_obj.get\_product\_id()

if product\_id not in self.\_\_product\_id\_set:

self.\_\_product\_list.append(product\_obj)

self.\_\_product\_id\_set.add(product\_id)

def read\_product\_fromSet(self, product\_id):

if product\_id in self.\_\_product\_id\_set:

for product in self.\_\_product\_list:

if product.get\_product\_id() == product\_id:

return product

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

def update\_product\_toSet(self, product\_id, new\_name, new\_price):

product\_obj = self.read\_product\_fromSet(product\_id)

product\_obj.set\_name(new\_name)

product\_obj.set\_price(new\_price)

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

if product\_id in self.\_\_product\_id\_set:

for product in self.\_\_product\_list:

if product.get\_product\_id() == product\_id:

self.\_\_product\_list.remove(product)

self.\_\_product\_id\_set.remove(product\_id)

return

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

Package Name: ShopException

File Name: shopexception.py

class ProductNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Product not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: ShopMain

File Name: MainModule.py

import logging

from Set.Q20.ProductDTO.product import Product

from Set.Q20.ShopDTO.shop import Shop

from Set.Q20.ShopException.shopexception import ProductNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="shop\_inventory.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

shop1 = Shop("S01", "Croma")

shop2 = Shop("S02", "HomeCentre")

s1\_p1 = Product("P101", "Laptop", 70000)

s1\_p2 = Product("P102", "Smartphone", 25000)

s1\_p3 = Product("P103", "Tablet", 15000)

s2\_p1 = Product("P201", "Mixer", 3500)

s2\_p2 = Product("P202", "Cooker", 2200)

s2\_p3 = Product("P203", "Refrigerator", 42000)

for product in [s1\_p1, s1\_p2, s1\_p3]:

shop1.add\_product\_toSet(product)

for product in [s2\_p1, s2\_p2, s2\_p3]:

shop2.add\_product\_toSet(product)

shops = [shop1, shop2]

for shop in shops:

logging.info(f"Shop ID: {shop.get\_shop\_id()} | Shop Name: {shop.get\_shop\_name()}")

for product in shop.get\_product\_list():

logging.info(product)

try:

shop1.update\_product\_toSet("P103", "Android Tablet", 17000)

logging.info("Shop1: Product P103 is updated.")

except ProductNotFoundError as e:

logging.error(e)

try:

shop1.remove\_product\_fromSet("P102")

logging.warning("Shop1: Product P102 is removed.")

except ProductNotFoundError as e:

logging.error(e)

logging.info(f"Updated list for Shop ID: {shop1.get\_shop\_id()} | Shop Name: {shop1.get\_shop\_name()}")

for product in shop1.get\_product\_list():

logging.info(product)

Directory Name: Tuple

Package Name: ProductDTO

File Name: product.py

class Product(object):

def \_\_init\_\_(self, product\_id, name, price):

self.\_\_product\_id = product\_id

self.\_\_name = name

self.\_\_price = price

def get\_product\_id(self):

return self.\_\_product\_id

def get\_name(self):

return self.\_\_name

def get\_price(self):

return self.\_\_price

def set\_name(self, name):

self.\_\_name = name

def set\_price(self, price):

self.\_\_price = price

def del\_product\_id(self):

del self.\_\_product\_id

def del\_name(self):

del self.\_\_name

def del\_price(self):

del self.\_\_price

def to\_tuple(self):

return (self.\_\_product\_id, self)

def \_\_str\_\_(self):

return f"Product ID: {self.\_\_product\_id} | Name: {self.\_\_name} | Price: Rs.{self.\_\_price}"

Package Name: ShopDTO

File Name: shop.py

from Tuple.Q20.ShopException.shopexception import ProductNotFoundError

class Shop(object):

def \_\_init\_\_(self, shop\_id, shop\_name):

self.\_\_shop\_id = shop\_id

self.\_\_shop\_name = shop\_name

self.\_\_product\_tuple = ()

def \_\_str\_\_(self):

result = f"Shop ID: {self.\_\_shop\_id} | Shop Name: {self.\_\_shop\_name}\nProducts:\n"

result += "\n".join(str(product[1]) for product in self.\_\_product\_tuple)

return result

def get\_shop\_id(self):

return self.\_\_shop\_id

def get\_shop\_name(self):

return self.\_\_shop\_name

def get\_product\_list(self):

return [product[1] for product in self.\_\_product\_tuple]

def set\_shop\_id(self, value):

self.\_\_shop\_id = value

def set\_shop\_name(self, value):

self.\_\_shop\_name = value

def set\_product\_tuple(self, value):

self.\_\_product\_tuple = tuple((p.get\_product\_id(), p) for p in value)

def add\_product\_toTuple(self, product\_obj):

self.\_\_product\_tuple += (product\_obj.to\_tuple(),)

def read\_product\_fromTuple(self, product\_id):

for pid, product in self.\_\_product\_tuple:

if pid == product\_id:

return product

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

def update\_product\_toTuple(self, product\_id, new\_name, new\_price):

updated = False

updated\_list = []

for pid, product in self.\_\_product\_tuple:

if pid == product\_id:

product.set\_name(new\_name)

product.set\_price(new\_price)

updated = True

updated\_list.append((pid, product))

if not updated:

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

self.\_\_product\_tuple = tuple(updated\_list)

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

found = False

filtered = []

for pid, product in self.\_\_product\_tuple:

if pid != product\_id:

filtered.append((pid, product))

else:

found = True

if not found:

raise ProductNotFoundError(f"Product with ID {product\_id} not found.")

self.\_\_product\_tuple = tuple(filtered)

Package Name: ShopException

File Name: shopexception.py

class ProductNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Product not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: ShopMain

File Name: MainModule.py

import logging

from Tuple.Q20.ProductDTO.product import Product

from Tuple.Q20.ShopDTO.shop import Shop

from Tuple.Q20.ShopException.shopexception import ProductNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="shop\_inventory.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

shop1 = Shop("S01", "Croma")

shop2 = Shop("S02", "HomeCentre")

s1\_p1 = Product("P101", "Laptop", 70000)

s1\_p2 = Product("P102", "Smartphone", 25000)

s1\_p3 = Product("P103", "Tablet", 15000)

s2\_p1 = Product("P201", "Mixer", 3500)

s2\_p2 = Product("P202", "Cooker", 2200)

s2\_p3 = Product("P203", "Refrigerator", 42000)

for product in [s1\_p1, s1\_p2, s1\_p3]:

shop1.add\_product\_toTuple(product)

for product in [s2\_p1, s2\_p2, s2\_p3]:

shop2.add\_product\_toTuple(product)

shops = [shop1, shop2]

for shop in shops:

logging.info(f"Shop ID: {shop.get\_shop\_id()} | Shop Name: {shop.get\_shop\_name()}")

for product in shop.get\_product\_list():

logging.info(product)

try:

shop1.update\_product\_toTuple("P103", "Android Tablet", 17000)

logging.info("Shop1: Product P103 is updated.")

except ProductNotFoundError as e:

logging.error(e)

try:

shop1.remove\_product\_fromTuple("P102")

logging.warning("Shop1: Product P102 is removed.")

except ProductNotFoundError as e:

logging.error(e)

logging.info(f"Updated list for Shop ID: {shop1.get\_shop\_id()} | Shop Name: {shop1.get\_shop\_name()}")

for product in shop1.get\_product\_list():

logging.info(product)

Company has employees, implement the requirement using collections

Directory Name: List

Sub-directory Name: Q21

Package Name: EmployeeDTO

File Name: employee.py

class Employee(object):

def \_\_init\_\_(self, emp\_id, name, salary):

self.\_\_emp\_id = emp\_id

self.\_\_name = name

self.\_\_salary = salary

def get\_emp\_id(self):

return self.\_\_emp\_id

def get\_name(self):

return self.\_\_name

def get\_salary(self):

return self.\_\_salary

def set\_name(self, name):

self.\_\_name = name

def set\_salary(self, salary):

self.\_\_salary = salary

def del\_emp\_id(self):

del self.\_\_emp\_id

def del\_name(self):

del self.\_\_name

def del\_salary(self):

del self.\_\_salary

def \_\_str\_\_(self):

return f"Emp ID: {self.\_\_emp\_id} | Name: {self.\_\_name} | Salary: Rs.{self.\_\_salary}"

Package Name: CompanyDTO

File Name: company.py

from List.Q21.CompanyException.companyexception import EmployeeNotFoundError

class Company(object):

def \_\_init\_\_(self, company\_id, company\_name):

self.\_\_company\_id = company\_id

self.\_\_company\_name = company\_name

self.\_\_employee\_list = []

def \_\_str\_\_(self):

result = f"Company ID: {self.\_\_company\_id} | Company Name: {self.\_\_company\_name}\nEmployees:\n"

result += "\n".join(str(emp) for emp in self.\_\_employee\_list)

return result

def get\_company\_id(self):

return self.\_\_company\_id

def get\_company\_name(self):

return self.\_\_company\_name

def get\_employee\_list(self):

return self.\_\_employee\_list

def set\_company\_id(self, value):

self.\_\_company\_id = value

def set\_company\_name(self, value):

self.\_\_company\_name = value

def set\_employee\_list(self, value):

self.\_\_employee\_list = value

def add\_emp\_toList(self, emp\_id):

self.\_\_employee\_list.append(emp\_id)

def read\_emp\_fromList(self, emp\_id):

for emp in self.\_\_employee\_list:

if emp.get\_emp\_id() == emp\_id:

return emp

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

def update\_emp\_toList(self, emp\_id, new\_name, new\_salary):

emp = self.read\_emp\_fromList(emp\_id)

emp.set\_name(new\_name)

emp.set\_salary(new\_salary)

def remove\_emp\_fromList(self, emp\_id):

emp = self.read\_emp\_fromList(emp\_id)

self.\_\_employee\_list.remove(emp)

Package Name: CompanyException

File Name: companyexception.py

class EmployeeNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Employee not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: CompanyMain

File Name: MainModule.py

import logging

from List.Q21.EmployeeDTO.employee import Employee

from List.Q21.CompanyDTO.company import Company

from List.Q21.CompanyException.companyexception import EmployeeNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="company\_employee.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

company = Company("C01", "HTC")

e1 = Employee("E101", "Amy", 50000)

e2 = Employee("E102", "Brad", 60000)

e3 = Employee("E103", "Caroline", 55000)

for emp in [e1, e2, e3]:

company.add\_emp\_toList(emp)

logging.info(f"Company ID: {company.get\_company\_id()} | Company Name: {company.get\_company\_name()}")

logging.info("List of all employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

try:

company.update\_emp\_toList("E103", "Charles", 58000)

logging.info("Employee E103 updated.")

except EmployeeNotFoundError as e:

logging.error(e)

try:

company.remove\_emp\_fromList("E102")

logging.warning("Employee E102 removed.")

except EmployeeNotFoundError as e:

logging.error(e)

logging.info("Final list of employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

Directory Name: Dictionary

Sub-directory Name: Q21

Package Name: EmployeeDTO

File Name: employee.py

class Employee(object):

def \_\_init\_\_(self, emp\_id, name, salary):

self.\_\_emp\_id = emp\_id

self.\_\_name = name

self.\_\_salary = salary

def get\_emp\_id(self):

return self.\_\_emp\_id

def get\_name(self):

return self.\_\_name

def get\_salary(self):

return self.\_\_salary

def set\_name(self, name):

self.\_\_name = name

def set\_salary(self, salary):

self.\_\_salary = salary

def del\_emp\_id(self):

del self.\_\_emp\_id

def del\_name(self):

del self.\_\_name

def del\_salary(self):

del self.\_\_salary

def \_\_str\_\_(self):

return f"Emp ID: {self.\_\_emp\_id} | Name: {self.\_\_name} | Salary: Rs.{self.\_\_salary}"

Package Name: CompanyDTO

File Name: company.py

from Dictionary.Q21.CompanyException.companyexception import EmployeeNotFoundError

class Company(object):

def \_\_init\_\_(self, company\_id, company\_name):

self.\_\_company\_id = company\_id

self.\_\_company\_name = company\_name

self.\_\_employee\_dict = {}

def \_\_str\_\_(self):

result = f"Company ID: {self.\_\_company\_id} | Company Name: {self.\_\_company\_name}\nEmployees:\n"

result += "\n".join(str(emp) for emp in self.\_\_employee\_dict.values())

return result

def get\_company\_id(self):

return self.\_\_company\_id

def get\_company\_name(self):

return self.\_\_company\_name

def get\_employee\_list(self):

return list(self.\_\_employee\_dict.values())

def set\_company\_id(self, value):

self.\_\_company\_id = value

def set\_company\_name(self, value):

self.\_\_company\_name = value

def set\_employee\_list(self, value):

self.\_\_employee\_dict = value

def add\_emp\_toDict(self, emp):

emp\_id = emp.get\_emp\_id()

self.\_\_employee\_dict[emp\_id] = emp

def read\_emp\_fromDict(self, emp\_id):

if emp\_id in self.\_\_employee\_dict:

return self.\_\_employee\_dict[emp\_id]

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

def update\_emp\_toDict(self, emp\_id, new\_name, new\_salary):

emp = self.read\_emp\_fromDict(emp\_id)

emp.set\_name(new\_name)

emp.set\_salary(new\_salary)

def remove\_emp\_fromDict(self, emp\_id):

if emp\_id in self.\_\_employee\_dict:

del self.\_\_employee\_dict[emp\_id]

else:

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

Package Name: CompanyException

File Name: companyexception.py

class EmployeeNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Employee not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: CompanyMain

File Name: MainModule.py

import logging

from Dictionary.Q21.EmployeeDTO.employee import Employee

from Dictionary.Q21.CompanyDTO.company import Company

from Dictionary.Q21.CompanyException.companyexception import EmployeeNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="company\_employee.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

company = Company("C01", "HTC")

e1 = Employee("E101", "Amy", 50000)

e2 = Employee("E102", "Brad", 60000)

e3 = Employee("E103", "Caroline", 55000)

for emp in [e1, e2, e3]:

company.add\_emp\_toDict(emp)

logging.info(f"Company ID: {company.get\_company\_id()} | Company Name: {company.get\_company\_name()}")

logging.info("List of all employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

try:

company.update\_emp\_toDict("E103", "Charles", 58000)

logging.info("Employee E103 updated.")

except EmployeeNotFoundError as e:

logging.error(e)

try:

company.remove\_emp\_fromDict("E102")

logging.warning("Employee E102 removed.")

except EmployeeNotFoundError as e:

logging.error(e)

logging.info("Final list of employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

Directory Name: Set

Sub-directory Name: Q21

Package Name: EmployeeDTO

File Name: employee.py

class Employee(object):

def \_\_init\_\_(self, emp\_id, name, salary):

self.\_\_emp\_id = emp\_id

self.\_\_name = name

self.\_\_salary = salary

def get\_emp\_id(self):

return self.\_\_emp\_id

def get\_name(self):

return self.\_\_name

def get\_salary(self):

return self.\_\_salary

def set\_name(self, name):

self.\_\_name = name

def set\_salary(self, salary):

self.\_\_salary = salary

def del\_emp\_id(self):

del self.\_\_emp\_id

def del\_name(self):

del self.\_\_name

def del\_salary(self):

del self.\_\_salary

def \_\_str\_\_(self):

return f"Emp ID: {self.\_\_emp\_id} | Name: {self.\_\_name} | Salary: Rs.{self.\_\_salary}"

Package Name: CompanyDTO

File Name: company.py

from Set.Q21.CompanyException.companyexception import EmployeeNotFoundError

class Company(object):

def \_\_init\_\_(self, company\_id, company\_name):

self.\_\_company\_id = company\_id

self.\_\_company\_name = company\_name

self.\_\_employee\_set = set()

def \_\_str\_\_(self):

result = f"Company ID: {self.\_\_company\_id} | Company Name: {self.\_\_company\_name}\nEmployees:\n"

result += "\n".join(str(emp) for emp in self.\_\_employee\_set)

return result

def get\_company\_id(self):

return self.\_\_company\_id

def get\_company\_name(self):

return self.\_\_company\_name

def get\_employee\_list(self):

return list(self.\_\_employee\_set)

def set\_company\_id(self, value):

self.\_\_company\_id = value

def set\_company\_name(self, value):

self.\_\_company\_name = value

def set\_employee\_set(self, value):

self.\_\_employee\_set = value

def add\_emp\_toSet(self, emp):

self.\_\_employee\_set.add(emp)

def read\_emp\_fromSet(self, emp\_id):

for emp in self.\_\_employee\_set:

if emp.get\_emp\_id() == emp\_id:

return emp

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

def update\_emp\_toSet(self, emp\_id, new\_name, new\_salary):

emp = self.read\_emp\_fromSet(emp\_id)

emp.set\_name(new\_name)

emp.set\_salary(new\_salary)

def remove\_emp\_fromSet(self, emp\_id):

emp\_to\_remove = None

for emp in self.\_\_employee\_set:

if emp.get\_emp\_id() == emp\_id:

emp\_to\_remove = emp

break

if emp\_to\_remove:

self.\_\_employee\_set.remove(emp\_to\_remove)

else:

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

Package Name: CompanyException

File Name: companyexception.py

class EmployeeNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Employee not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: CompanyMain

File Name: MainModule.py

import logging

from Set.Q21.EmployeeDTO.employee import Employee

from Set.Q21.CompanyDTO.company import Company

from Set.Q21.CompanyException.companyexception import EmployeeNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="company\_employee.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

company = Company("C01", "HTC")

e1 = Employee("E101", "Amy", 50000)

e2 = Employee("E102", "Brad", 60000)

e3 = Employee("E103", "Caroline", 55000)

for emp in [e1, e2, e3]:

company.add\_emp\_toSet(emp)

logging.info(f"Company ID: {company.get\_company\_id()} | Company Name: {company.get\_company\_name()}")

logging.info("List of all employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

try:

company.update\_emp\_toSet("E103", "Charles", 58000)

logging.info("Employee E103 updated.")

except EmployeeNotFoundError as e:

logging.error(e)

try:

company.remove\_emp\_fromSet("E102")

logging.warning("Employee E102 removed.")

except EmployeeNotFoundError as e:

logging.error(e)

logging.info("Final list of employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

Directory Name: Tuple

Sub-directory Name: Q21

Package Name: EmployeeDTO

File Name: employee.py

class Employee(object):

def \_\_init\_\_(self, emp\_id, name, salary):

self.\_\_emp\_id = emp\_id

self.\_\_name = name

self.\_\_salary = salary

def get\_emp\_id(self):

return self.\_\_emp\_id

def get\_name(self):

return self.\_\_name

def get\_salary(self):

return self.\_\_salary

def set\_name(self, name):

self.\_\_name = name

def set\_salary(self, salary):

self.\_\_salary = salary

def del\_emp\_id(self):

del self.\_\_emp\_id

def del\_name(self):

del self.\_\_name

def del\_salary(self):

del self.\_\_salary

def \_\_str\_\_(self):

return f"Emp ID: {self.\_\_emp\_id} | Name: {self.\_\_name} | Salary: Rs.{self.\_\_salary}"

Package Name: CompanyDTO

File Name: company.py

from Tuple.Q21.CompanyException.companyexception import EmployeeNotFoundError

class Company(object):

def \_\_init\_\_(self, company\_id, company\_name):

self.\_\_company\_id = company\_id

self.\_\_company\_name = company\_name

self.\_\_employee\_tuple = () # empty tuple initially

def \_\_str\_\_(self):

result = f"Company ID: {self.\_\_company\_id} | Company Name: {self.\_\_company\_name}\nEmployees:\n"

result += "\n".join(str(emp) for emp in self.\_\_employee\_tuple)

return result

def get\_company\_id(self):

return self.\_\_company\_id

def get\_company\_name(self):

return self.\_\_company\_name

def get\_employee\_list(self):

return list(self.\_\_employee\_tuple) # convert tuple to list for easier use

def set\_company\_id(self, value):

self.\_\_company\_id = value

def set\_company\_name(self, value):

self.\_\_company\_name = value

def set\_employee\_tuple(self, value):

self.\_\_employee\_tuple = value

def add\_emp\_toTuple(self, emp):

self.\_\_employee\_tuple = self.\_\_employee\_tuple + (emp,)

def read\_emp\_fromTuple(self, emp\_id):

for emp in self.\_\_employee\_tuple:

if emp.get\_emp\_id() == emp\_id:

return emp

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

def update\_emp\_toTuple(self, emp\_id, new\_name, new\_salary):

found = False

new\_emps = []

for emp in self.\_\_employee\_tuple:

if emp.get\_emp\_id() == emp\_id:

emp.set\_name(new\_name)

emp.set\_salary(new\_salary)

found = True

new\_emps.append(emp)

if not found:

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

self.\_\_employee\_tuple = tuple(new\_emps)

def remove\_emp\_fromTuple(self, emp\_id):

new\_emps = tuple(emp for emp in self.\_\_employee\_tuple if emp.get\_emp\_id() != emp\_id)

if len(new\_emps) == len(self.\_\_employee\_tuple):

raise EmployeeNotFoundError(f"Employee with ID {emp\_id} not found.")

self.\_\_employee\_tuple = new\_emps

Package Name: CompanyException

File Name: companyexception.py

class EmployeeNotFoundError(Exception):

def \_\_init\_\_(self, errormessage="Employee not found"):

self.\_\_errormessage = errormessage

def \_\_str\_\_(self):

return self.\_\_errormessage

Package Name: CompanyMain

File Name: MainModule.py

import logging

from Tuple.Q21.EmployeeDTO.employee import Employee

from Tuple.Q21.CompanyDTO.company import Company

from Tuple.Q21.CompanyException.companyexception import EmployeeNotFoundError

if \_\_name\_\_ == "\_\_main\_\_":

logging.basicConfig(filename="company\_employee.log", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")

company = Company("C01", "HTC")

e1 = Employee("E101", "Amy", 50000)

e2 = Employee("E102", "Brad", 60000)

e3 = Employee("E103", "Caroline", 55000)

for emp in [e1, e2, e3]:

company.add\_emp\_toTuple(emp)

logging.info(f"Company ID: {company.get\_company\_id()} | Company Name: {company.get\_company\_name()}")

logging.info("List of all employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

try:

company.update\_emp\_toTuple("E103", "Charles", 58000)

logging.info("Employee E103 updated.")

except EmployeeNotFoundError as e:

logging.error(e)

try:

company.remove\_emp\_fromTuple("E102")

logging.warning("Employee E102 removed.")

except EmployeeNotFoundError as e:

logging.error(e)

logging.info("Final list of employees:")

for emp in company.get\_employee\_list():

logging.info(emp)

File Handling

Filter Products by Category and Write to New File

Write a Python program to read product.xlsx and filter the products based on a specified category (e.g., "Electronics"). Write the filtered products into a new file, filtered\_products.xlsx. If the specified category doesn't exist in the dataset, raise a custom exception CategoryNotFoundException and log the error.

File Name: exceptions.py

class CategoryNotFoundException(Exception):

pass

File Name: filterproducts.py

import openpyxl

import logging

from openpyxl import load\_workbook, Workbook

from exceptions import CategoryNotFoundException

# Configure logging to write errors/info into a log file named 'productfilter.log'

logging.basicConfig(

filename="productfilter.log",

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s"

)

# Load the Excel workbook from the given file path

workbook = load\_workbook(r"C:\Users\Tina L\PycharmProjects\PythonWork-14.05.2025\Q1\filter\product.xlsx")

# Select the sheet named "Products"

sheet = workbook["Products"]

# Read and store the column headers from the first row

header = []

for i in range(1, sheet.max\_column + 1):

cellValue = sheet.cell(row=1, column=i).value

header.append(cellValue)

# Try to find the index of the "Category" column

try:

catIndex = header.index("Category") + 1 # openpyxl columns are 1-indexed

except ValueError:

raise ValueError("'Category' column does not exist in 'product.xlsx'") # Raise error if not found

# Define the category to filter

catName = "Furniture"

# List to hold filtered rows, starting with the header

filteredRows = []

filteredRows.append(header)

# Iterate over all rows and filter rows that match the specified category

for i in range(2, sheet.max\_row + 1): # Start from row 2 to skip header

catCell = sheet.cell(row=i, column=catIndex).value

if catCell == catName:

rowData = []

for j in range(1, sheet.max\_column + 1):

cellValue = sheet.cell(row=i, column=j).value

rowData.append(cellValue)

filteredRows.append(rowData)

# Raise an exception if no matching products are found

if len(filteredRows) == 1:

raise CategoryNotFoundException(f"No products are found in the category {catName}")

# Create a new workbook and write the filtered data to it

newWorkbook = Workbook()

newSheet = newWorkbook.active

newSheet.title = "Filtered products"

# Append all filtered rows into the new sheet

for i in filteredRows:

newSheet.append(i)

# Save the new Excel file

newWorkbook.save("filtered\_products.xlsx")

# Log info message about successful write

logging.info(f"Filtered products written to 'filtered\_products.xlsx'.")

Append New Products to the File

Allow users to add a new product (Product ID, Name, Category, Price, Stock) to product.xlsx. If the product ID already exists, raise a DuplicateProductException.

File Name: exceptions.py

class DuplicateProductException(Exception):

pass

File Name: appendproducts.py

import openpyxl

import logging

from openpyxl import load\_workbook

from exceptions import DuplicateProductException

# Setup logging

logging.basicConfig(filename=" productappend.log", level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s")

# Load workbook and define target file/sheet

file\_path = "product.xlsx"

workbook = load\_workbook(file\_path)

# Create or get 'NewProducts' sheet

sheet\_name = "NewProducts"

if sheet\_name in workbook.sheetnames:

sheet = workbook[sheet\_name]

else:

sheet = workbook.create\_sheet(title=sheet\_name)

sheet.append(["Product ID", "Name", "Category", "Price", "Stock"]) # Header

# Read header

header = []

for col in range(1, sheet.max\_column + 1):

header.append(sheet.cell(row=1, column=col).value)

# Get index of 'Product ID' column

try:

product\_id\_index = header.index("Product ID") + 1

except ValueError:

raise ValueError("'Product ID' column not found in the sheet.")

# New product details

new\_product\_id = "P012"

new\_name = "Smartwatch"

new\_category = "Electronics"

new\_price = 6000

new\_stock = 20

# Check for duplicate Product ID

for row in range(2, sheet.max\_row + 1):

existing\_id = sheet.cell(row=row, column=product\_id\_index).value

if existing\_id == new\_product\_id:

error\_message = f"Product ID '{new\_product\_id}' already exists in '{sheet\_name}'."

logging.error(error\_message)

raise DuplicateProductException(error\_message)

# Create new row based on header

new\_row = []

for column\_name in header:

if column\_name == "Product ID":

new\_row.append(new\_product\_id)

elif column\_name == "Name":

new\_row.append(new\_name)

elif column\_name == "Category":

new\_row.append(new\_category)

elif column\_name == "Price":

new\_row.append(new\_price)

elif column\_name == "Stock":

new\_row.append(new\_stock)

else:

new\_row.append(None)

# Append new row to new sheet

sheet.append(new\_row)

# Save changes

workbook.save(file\_path)

logging.info(f"Product '{new\_name}' added successfully to sheet '{sheet\_name}' in '{file\_path}'.")

Read Employee Details and Validate Salary

Read employee.xlsx and check if any employee has a salary below the minimum threshold (e.g., 10000). Raise a LowSalaryException for such cases and log the errors into error\_log.txt.

File Name: exceptions.py

class LowSalaryException(Exception):

pass

File Name: validatesalary.py

import openpyxl

import logging

from openpyxl import load\_workbook

from exceptions import LowSalaryException

# Setup logging

logging.basicConfig(filename="salaryvalidate.log", level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s")

# Load the workbook and worksheet

file\_path = "employee.xlsx"

workbook = load\_workbook(file\_path)

sheet = workbook["Employees"]

# Read the header

header = []

for col in range(1, sheet.max\_column + 1):

cell\_value = sheet.cell(row=1, column=col).value

header.append(cell\_value)

# Identify Salary column index

try:

salary\_index = header.index("Salary") + 1

except ValueError:

raise ValueError("'Salary' column not found in the Excel sheet.")

# Threshold salary

min\_salary = 10000

# Loop through each row and validate salary

for row in range(2, sheet.max\_row + 1):

emp\_id = sheet.cell(row=row, column=1).value

name = sheet.cell(row=row, column=2).value

salary = sheet.cell(row=row, column=salary\_index).value

if salary is not None and salary < min\_salary:

error\_msg = f"Low salary for {emp\_id} ({name}): Rs.{salary}"

logging.error(error\_msg)

raise LowSalaryException(error\_msg)

Update Customer Email Address

Read customer.xlsx and update the email address for a given Customer ID. If the Customer ID does not exist, raise a CustomerNotFoundException.

File Name: exceptions.py

class CustomerNotFoundException(Exception):

pass

File Name: updatecustomer.py

import openpyxl

import logging

from openpyxl import load\_workbook

from exceptions import CustomerNotFoundException

# Configure logging

logging.basicConfig(filename="customerupdate.log", level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s")

# Load the workbook and sheet

file\_path = "customer.xlsx"

workbook = load\_workbook(file\_path)

sheet = workbook["Customers"] # Ensure the sheet is named 'Customers'

# Read header row to find indexes

header = []

for col in range(1, sheet.max\_column + 1):

header.append(sheet.cell(row=1, column=col).value)

# Identify the index of 'Customer ID' and 'Email'

try:

cust\_id\_index = header.index("Customer ID") + 1

email\_index = header.index("Email") + 1

except ValueError as e:

raise ValueError("Required column is missing in the Excel sheet.") from e

# Input: Customer ID and new Email

target\_id = "C005"

new\_email = "ethan.newemail@example.com"

# Track if ID found

found = False

# Search for the customer and update email

for row in range(2, sheet.max\_row + 1):

current\_id = sheet.cell(row=row, column=cust\_id\_index).value

if current\_id == target\_id:

sheet.cell(row=row, column=email\_index).value = new\_email

found = True

logging.info(f"Updated email for Customer ID {target\_id} to {new\_email}")

break

# If ID not found, raise exception

if not found:

error\_msg = f"Customer ID '{target\_id}' not found in the dataset."

logging.error(error\_msg)

raise CustomerNotFoundException(error\_msg)

# Save changes

workbook.save(file\_path)

Calculate Total Stock Value

Read product.xlsx, calculate the total stock value (Price \* Qty) for each product, and store the results in a new file stock\_value.xlsx. If a product has negative qty, raise a NegativeStockException.

File Name: exceptions.py

class NegativeStockException(Exception):

pass

File Name: calculatestockvalue.py

import openpyxl

import logging

from openpyxl import load\_workbook, Workbook

from exceptions import NegativeStockException

# Setup logging

logging.basicConfig(

filename="stockvalue.log",

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s"

)

# Load the existing workbook and sheet

workbook = load\_workbook("product.xlsx")

sheet = workbook["Products"]

# Read header row and find column indexes

header = []

for col in range(1, sheet.max\_column + 1):

header.append(sheet.cell(row=1, column=col).value)

try:

price\_index = header.index("Price") + 1

stock\_index = header.index("Stock") + 1

except ValueError:

raise ValueError("Required 'Price' or 'Stock' column not found.")

# Create a new workbook to save the output

new\_wb = Workbook()

new\_sheet = new\_wb.active

new\_sheet.title = "Stock Value"

# Write header to new file

new\_sheet.append(header + ["Total Stock Value"])

# Process each row

for row in range(2, sheet.max\_row + 1):

product\_id = sheet.cell(row=row, column=1).value

price = sheet.cell(row=row, column=price\_index).value

stock = sheet.cell(row=row, column=stock\_index).value

if stock is not None and stock < 0:

error\_msg = f"Negative stock found for Product ID '{product\_id}': {stock}"

logging.error(error\_msg)

raise NegativeStockException(error\_msg)

total\_value = price \* stock if price is not None and stock is not None else 0

# Read original row data

row\_data = []

for col in range(1, sheet.max\_column + 1):

row\_data.append(sheet.cell(row=row, column=col).value)

# Append total stock value

row\_data.append(total\_value)

new\_sheet.append(row\_data)

# Save the new workbook

new\_wb.save("stock\_value.xlsx")

logging.info("Stock value calculation completed successfully.")

Merge Employee and Customer Data

Read employee.xlsx and customer.xlsx and merge records where employees are also customers (matching by email). Save the merged data to employee\_customer.xlsx. Handle missing email entries with EmailNotFoundException.

File Name: exceptions.py

class EmailNotFoundException(Exception):

pass

File Name:

import openpyxl

import logging

from openpyxl import load\_workbook, Workbook

from exceptions import EmailNotFoundException

# Setup logging

logging.basicConfig(

filename="employeecustomermerge.log",

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s"

)

# Load workbooks

emp\_wb = load\_workbook("employee.xlsx")

cust\_wb = load\_workbook("customer.xlsx")

# Access the first sheets

emp\_sheet = emp\_wb.active

cust\_sheet = cust\_wb.active

# Read headers

emp\_headers = [emp\_sheet.cell(row=1, column=i).value for i in range(1, emp\_sheet.max\_column + 1)]

cust\_headers = [cust\_sheet.cell(row=1, column=i).value for i in range(1, cust\_sheet.max\_column + 1)]

try:

emp\_email\_col = emp\_headers.index("Email") + 1

cust\_email\_col = cust\_headers.index("Email") + 1

except ValueError:

raise ValueError("Missing 'Email' column in one of the sheets.")

# Build a lookup dictionary for customers

customer\_dict = {}

for i in range(2, cust\_sheet.max\_row + 1):

email = cust\_sheet.cell(row=i, column=cust\_email\_col).value

if email:

customer\_data = [cust\_sheet.cell(row=i, column=j).value for j in range(1, cust\_sheet.max\_column + 1)]

customer\_dict[email.strip().lower()] = customer\_data

# Prepare new workbook for merged data

merged\_wb = Workbook()

merged\_sheet = merged\_wb.active

merged\_sheet.title = "Employee-Customer"

# Create combined header

merged\_header = emp\_headers + [h for h in cust\_headers if h not in emp\_headers]

merged\_sheet.append(merged\_header)

# Merge matching records

for i in range(2, emp\_sheet.max\_row + 1):

emp\_email = emp\_sheet.cell(row=i, column=emp\_email\_col).value

if not emp\_email:

msg = f"Missing email in employee row {i}"

logging.error(msg)

raise EmailNotFoundException(msg)

emp\_email\_key = emp\_email.strip().lower()

if emp\_email\_key in customer\_dict:

emp\_data = [emp\_sheet.cell(row=i, column=j).value for j in range(1, emp\_sheet.max\_column + 1)]

cust\_data = customer\_dict[emp\_email\_key]

# Add only non-duplicate customer fields

cust\_extra\_data = [cust\_data[j] for j in range(len(cust\_headers)) if cust\_headers[j] not in emp\_headers]

merged\_row = emp\_data + cust\_extra\_data

merged\_sheet.append(merged\_row)

# Save merged workbook

merged\_wb.save("employee\_customer.xlsx")

logging.info("Merged employee and customer data written to 'employee\_customer.xlsx'")

Generate a Sales Report from Multiple Sheets

Assume sales.xlsx contains multiple sheets with sales data. Write a Python program to read all sheets and generate a consolidated sales report. Handle the case where a sheet is missing using SheetNotFoundException.

File Name: exceptions.py

class SheetNotFoundException(Exception):

pass

File Name: generatesalesreport.py

import openpyxl

import logging

from openpyxl import load\_workbook, Workbook

from exceptions import SheetNotFoundException

# Logging configuration

logging.basicConfig(filename="salesreport.log", level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s")

# File path

file\_path = "sales.xlsx"

# Load workbook

try:

workbook = load\_workbook(file\_path)

except FileNotFoundError:

logging.error("File 'sales.xlsx' not found.")

raise

# Expected sheets

expected\_sheets = workbook.sheetnames

consolidated\_data = []

header\_written = False

# Iterate through each sheet

for sheet\_name in expected\_sheets:

if sheet\_name not in workbook.sheetnames:

error\_msg = f"Sheet '{sheet\_name}' is missing in workbook."

logging.error(error\_msg)

raise SheetNotFoundException(error\_msg)

sheet = workbook[sheet\_name]

logging.info(f"Processing sheet: {sheet\_name}")

# Read rows

for i in range(1, sheet.max\_row + 1):

row\_data = [sheet.cell(row=i, column=j).value for j in range(1, sheet.max\_column + 1)]

if i == 1:

if not header\_written:

consolidated\_data.append(row\_data)

header\_written = True

else:

consolidated\_data.append(row\_data)

# Write consolidated report

report\_wb = Workbook()

report\_sheet = report\_wb.active

report\_sheet.title = "Sales Report"

for row in consolidated\_data:

report\_sheet.append(row)

report\_wb.save("consolidated\_sales\_report.xlsx")

logging.info("Consolidated sales report saved as 'consolidated\_sales\_report.xlsx'.")

Delete Products with Zero Stock

Read product.xlsx and remove any product with zero qty. Save the updated file. If no such product exists, raise a NoZeroStockException.

File Name: exceptions.py

class NoZeroStockException(Exception):

pass

File Name: deletezerostockproducts.py

import openpyxl

import logging

from openpyxl import load\_workbook

from exceptions import NoZeroStockException

# Configure logging

logging.basicConfig(filename="zerostockdeletion.log", level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s")

# Load the workbook and access the "Products" sheet

file\_path = "product.xlsx"

workbook = load\_workbook(file\_path)

sheet = workbook["Products"]

# Identify the column index for 'Stock'

header = []

for col in range(1, sheet.max\_column + 1):

header.append(sheet.cell(row=1, column=col).value)

try:

stock\_index = header.index("Stock") + 1

except ValueError:

raise ValueError("'Stock' column not found in the sheet.")

# Track rows to delete

rows\_to\_delete = []

for row in range(2, sheet.max\_row + 1):

stock\_value = sheet.cell(row=row, column=stock\_index).value

if stock\_value == 0:

rows\_to\_delete.append(row)

if not rows\_to\_delete:

raise NoZeroStockException("No products with zero stock found.")

# Delete rows in reverse to avoid row shifting

for row in sorted(rows\_to\_delete, reverse=True):

sheet.delete\_rows(row)

# Save updated workbook

workbook.save("updated\_product.xlsx")

logging.info("Updated file saved as 'updated\_product.xlsx'.")

Employee Bonus Calculation

Read employee.xlsx, calculate a 10% bonus for employees earning above $5000, and save the updated data to bonus.xlsx. If an employee already has a bonus column, raise a BonusAlreadyExistsException.

File Name: exceptions.py

class BonusAlreadyExistsException(Exception):

pass

File Name: calculatebonus.py

import openpyxl

from openpyxl import load\_workbook, Workbook

from exceptions import BonusAlreadyExistsException

import logging

# Logging setup

logging.basicConfig(filename="bonus.log", level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s")

# Load employee.xlsx

file\_path = "employee.xlsx"

workbook = load\_workbook(file\_path)

sheet = workbook.active

# Extract header row

header = []

for col in range(1, sheet.max\_column + 1):

header.append(sheet.cell(row=1, column=col).value)

# Check if "Bonus" column already exists

if "Bonus" in header:

raise BonusAlreadyExistsException("'Bonus' column already exists in employee.xlsx.")

# Get index of 'Salary' column

try:

salary\_col\_index = header.index("Salary") + 1

except ValueError:

raise ValueError("'Salary' column not found in employee.xlsx.")

# Add "Bonus" to the header

bonus\_col\_index = len(header) + 1

sheet.cell(row=1, column=bonus\_col\_index).value = "Bonus"

# Calculate 10% bonus for salary > 5000

for row in range(2, sheet.max\_row + 1):

salary = sheet.cell(row=row, column=salary\_col\_index).value

if isinstance(salary, (int, float)) and salary > 5000:

bonus = round(salary \* 0.10, 2)

else:

bonus = 0

sheet.cell(row=row, column=bonus\_col\_index).value = bonus

# Save updated workbook

workbook.save("bonus.xlsx")

logging.info("Bonus calculation completed. Saved to 'bonus.xlsx'.")

Backup and Restore Customer Data

Create a backup of customer.xlsx as customer\_backup.xlsx. Allow restoring data from the backup when requested. If the backup file is missing, raise a BackupNotFoundException.

File Name: exceptions.py

class BackupNotFoundException(Exception):

pass

File Name: backuprestorecustomer.py

import os

import shutil

import logging

from exceptions import BackupNotFoundException

# Configure logging

logging.basicConfig(filename="customerbackuprestore.log", level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s")

# File paths

original\_file = "customer.xlsx"

backup\_file = "customer\_backup.xlsx"

# Choose mode

mode = input("Enter 'backup' to create backup or 'restore' to restore from backup: ").strip().lower()

if mode == "backup":

try:

shutil.copy(original\_file, backup\_file)

logging.info("Backup created successfully.")

print("Backup created as 'customer\_backup.xlsx'.")

except FileNotFoundError:

logging.error(f"{original\_file} not found.")

print("Original customer file not found.")

elif mode == "restore":

if not os.path.exists(backup\_file):

logging.error("Backup file not found.")

raise BackupNotFoundException("customer\_backup.xlsx not found. Cannot restore.")

shutil.copy(backup\_file, original\_file)

logging.info("Data restored from backup.")

logging.info("Data restored from 'customer\_backup.xlsx'.")

else:

logging.info("Invalid mode. Choose 'backup' or 'restore'.")