

```

#L1
#Q1
def calculate_area(length, width):
    return length * width

def categorize_property(area):
    if area < 1000:
        return "Small"
    elif area < 5000:
        return "Medium"
    else:
        return "Large"



def main():
    length = float(input("Enter the length of the property in meters"))
    width = float(input("Enter the width of the property in meters"))

    area = calculate_area(length, width)
    category = categorize_property(area)

    print("The area of the property is:", area, "square meters")
    print("This property is categorized as:", category)

if __name__ == "__main__":
    main()

```

  -----

```

KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-3-25a07f8f5f17> in <cell line: 25>()
    24
--> 25 if __name__ == "__main__":
    26     main()

----- 2 frames -----
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent, password)
    893         except KeyboardInterrupt:
    894             # re-raise KeyboardInterrupt, to truncate traceback
--> 895             raise KeyboardInterrupt("Interrupted by user") from None
    896     except Exception as e:
    897         self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user

```

Q2:

```
class HealthManager:
    def __init__(self, name, age, gender, weight, height):
        self.name = name
        self.age = age
        self.gender = gender
        self.weight = weight
        self.height = height

    def calculate_bmi(self):
        return self.weight / (self.height ** 2)

    def suggest_diet_plan(self):
        return "Sample diet plan: Include more fruits, vegetables, and whole grains in your diet."
    def suggest_fitness_plan(self):
        return "Sample fitness plan: Start with 30 minutes of moderate-intensity cardio exercises daily."

name = input("Enter your name: ")
age = int(input("Enter your age: "))
gender = input("Enter your gender (Male/Female): ")
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))
user = HealthManager(name, age, gender, weight, height)
bmi = user.calculate_bmi()
print("Your BMI is:", bmi)
if bmi < 18.5:
    print("Underweight")
elif 18.5 <= bmi < 25:
    print("Normal weight")
elif 25 <= bmi < 30:
    print("Overweight")
else:
    print("Obese")
print("Recommended diet plan:",
user.suggest_diet_plan())
print("Recommended fitness plan:", user.suggest_fitness_plan())
```

```

#Q3:
class SchoolManagementSystem:
    def __init__(self):
        self.students = {}

    def add_student(self, student_id, name):
        if student_id not in self.students:
            self.students[student_id] = {'name': name, 'grades': {}}
            print(f"Student {name} added successfully.")
        else:
            print(f"Student with ID {student_id} already exists.")

    def update_grade(self, student_id, subject, grade):
        if student_id in self.students:
            self.students[student_id]['grades'][subject] = grade
            print(f"Grade for {subject} updated successfully for student ID {student_id}.")
        else:
            print(f"Student with ID {student_id} not found.")

    def get_student_record(self, student_id):
        if student_id in self.students:
            student_info = self.students[student_id]
            print(f"Student ID: {student_id}")
            print(f"Name: {student_info['name']}")
            print("Grades:")
            for subject, grade in student_info['grades'].items():
                print(f"- {subject}: {grade}")
        else:
            print(f"Student with ID {student_id} not found.")

    def print_all_records(self):
        for student_id, student_info in self.students.items():
            print(f"Student ID: {student_id}")
            print(f"Name: {student_info['name']}")
            print("Grades:")
            for subject, grade in student_info['grades'].items():
                print(f"- {subject}: {grade}")
            print()

school_system = SchoolManagementSystem()
# Add students
school_system.add_student(1, "Alice")
school_system.add_student(2, "Bob")
# Update grades
school_system.update_grade(1, "Math", 85)
school_system.update_grade(1, "Science", 90)
school_system.update_grade(2, "Math", 75)
school_system.update_grade(2, "Science", 80)
# Retrieve student records
school_system.get_student_record(1)
school_system.get_student_record(2)
# Print all records
school_system.print_all_records()

```

```

#L2
#Q4:
def recommend_content():

    age = int(input("Please enter your age: "))
    if age < 13:
        print("Welcome to the Children's Collection!")
        print("Recommended movies: Finding Nemo, Toy Story, Frozen")
        print("Recommended TV shows: Peppa Pig, Paw Patrol, SpongeBob SquarePants")
    elif 13 <= age < 18:
        print("Welcome to the Teen Collection!")
        print("Recommended movies: Harry Potter series, The Hunger Games, Twilight")
        print("Recommended TV shows: Stranger Things, The Vampire Diaries, Riverdale")
    elif 18 <= age < 65:
        print("Welcome to the Adult Collection!")

        print("Recommended movies The Godfather, The Shawshank Redemption, Pulp Fiction")
        print("Recommended TV shows Breaking Bad, Game of Thrones, Friends")
    else:
        print("Welcome to the Senior Collection!")
        print("Recommended movies: The Bucket List, The Best Exotic Marigold Hotel, Up")
        print("Recommended TV shows: Grace and Frankie, The Crown, Downton Abbey")
    recommend_content()

#Q5:
def get_even_subscriber_ids(database):
    even_ids = []

    for subscriber_id in database:
        if subscriber_id % 2 == 0:
            even_ids.append(subscriber_id)
    return even_ids

subscriber_database = [101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
even_subscriber_ids = get_even_subscriber_ids(subscriber_database)
print("Even-numbered Subscriber IDs for the Promotional Email Campaign:")
print(even_subscriber_ids)

#Q6:
import hashlib
def generate_hash(password):
    return hashlib.sha256(password.encode()).hexdigest()
stored_password_hash = generate_hash("password123")
def check_password(password):
    return generate_hash(password) == stored_password_hash
def main():
    while True:
        entered_password = input("Enter your password to access sensitive financial documents: ")
        if check_password(entered_password):
            print("Access granted! You can now access the sensitive financial documents.")
            break
        else:
            print("Incorrect password. Please try again.")
main()

```

```
#Q7:
def analyze_customer_feedback(survey_data):
    total_score = 0

    num_responses = len(survey_data)

    for response in survey_data:
        total_score += response['satisfaction_score']

    average_score = total_score / num_responses

    print("Average satisfaction score:", average_score)

    if average_score < 3.5:
        print("Average satisfaction score is below expectation. Address common complaints.")
    elif average_score > 4.5:
        print("Average satisfaction score is high. Identify areas of strength.")
    else:
        print("Average satisfaction score is
within expectation. Look for areas of improvement.")
survey_data = [
    {'customer_id': 1, 'satisfaction_score': 4, 'comments': 'Great service!'},
    {'customer_id': 2, 'satisfaction_score': 3, 'comments': 'Room for improvement.'},
    {'customer_id': 3, 'satisfaction_score': 5, 'comments': 'Excellent experience!'},
    {'customer_id': 4, 'satisfaction_score': 2, 'comments': 'Poor communication.'},
    {'customer_id': 5, 'satisfaction_score': 4, 'comments': 'Satisfied overall.'}
]
analyze_customer_feedback(survey_data)
```

```
#Q8:
def count_vowels(comment):
    vowels = 'aeiouAEIOU'

    vowel_count = 0
    for char in comment:
        if char in vowels:
            vowel_count += 1
    return vowel_count
comment = "Wow, this is a great post! Thanks for sharing!"
num_vowels = count_vowels(comment)
print("Number of vowels:", num_vowels)
```

```
#Q9
import datetime

def generate_reminder(event, event_date, reminder_time):
    event_datetime = datetime.datetime.strptime(event_date,
'%Y-%m-%d')
    reminder_timedelta = datetime.timedelta(hours=reminder_time)
    # Calculate reminder datetime
    reminder_datetime = event_datetime - reminder_timedelta
    current_datetime = datetime.datetime.now()
    if reminder_datetime > current_datetime:
        time_until_reminder = reminder_datetime - current_datetime
        print(f"Reminder for '{event}' will be sent {time_until_reminder} before the event.")
    else:
        print("Event has already passed. No reminder will be sent.")
event = "Project Deadline"
event_date = "2024-05-01"
reminder_time = 24

generate_reminder(event, event_date, reminder_time)
```

Q10:

```
def calculate_loan_repayment(principal, interest_rate, num_years):  
    try:  
  
        interest_rate_decimal = interest_rate / 100  
        monthly_interest_rate = interest_rate_decimal / 12  
        num_payments = num_years * 12  
  
        # Calculate monthly payment  
        monthly_payment = (principal * monthly_interest_rate) / (1 - (1 + monthly_interest_rate) ** -num_payments)
```