# Computer Science Exam

## Question 16(a):

Consider the following Python program that calculates the total cost of a meal based on the price of the meal and the number of people sharing the meal:

```
# Question 16(a)
# Examination Number:

meal_price = float(input("Enter the price of the meal: "))
num_people = 2

total_cost = meal_price / num_people

print("The total cost per person is: ", total_cost)
```

A sample run of the program is displayed below – the user enters a value of 50 for the meal price and the program displays the resulting total cost per person.

```
Enter the price of the meal: 50
The total cost per person is:  25.0
```

Modify the program to do the following:
(i) Insert a comment to say 'read meal price' in the appropriate location in the program to show where the meal price is input.

(ii) Currently in the program, the value of the variable `num_people` is hard-coded to 2. Modify the program so that it prompts the user to enter the number of people sharing the meal. The value should be converted to an integer.

When the program is run, the output may look as follows:

```
Enter the price of the meal: 50
Enter the number of people sharing the meal: 4
The total cost per person is: 12.5
```

(iii) By using the function `round`, or otherwise, modify the program so that the value of the total cost per person displayed is rounded to two decimal places.

When the program is run, the output may look as follows:

```
Enter the price of the meal: 50
Enter the number of people sharing the meal: 3
The total cost per person is: 16.67
```

(iv) Incorporate the following function definition into your program and insert a line so that the function is called before the user enters any data.

```
def display_intro():
    print("Welcome to the Meal Cost Calculator!")
```

When the program is run, the output may look as follows:

```
Welcome to the Meal Cost Calculator!
Enter the price of the meal: 80
Enter the number of people sharing the meal: 5
The total cost per person is: 16.00
```

(v) Extend the program to handle the case where the user enters 0 for the number of people sharing the meal. If the user enters 0, the program should display an error message saying "Number of people cannot be zero." and terminate without performing the calculation.

When the program is run with 0 entered for the number of people, the output may look as follows:

```
Welcome to the Meal Cost Calculator!
Enter the price of the meal: 60
Enter the number of people sharing the meal: 0
Number of people cannot be zero.
```

(vi) Modify the program to calculate and display the tip amount per person in addition to the total cost per person. The tip percentage should be set to 15% of the meal price. The tip amount per person should be rounded to two decimal places.

When the program is run, the output may look as follows:

```
Welcome to the Meal Cost Calculator!
Enter the price of the meal: 100
Enter the number of people sharing the meal: 4
```

```
The total cost per person is: 25.00
The tip amount per person is: 3.75
```

Save and close your file before moving on to the next part.

## Question 16(b):

Consider the following Python program that determines the grade based on a student's score:

```python
# Question 16(b)
# Examination Number:

score = int(input("Enter the student's score: "))

if score ≥ 90:
    grade = 'H1'
elif score ≥ 80:
    grade = 'H2'
elif score ≥ 70:
    grade = 'H3'
elif score ≥ 60:
    grade = 'H4'
else:
    grade = 'H8'

print("The student's grade is:", grade)
```

A sample run of the program is displayed below – the user enters a score of 85 and the program displays the resulting grade.

```
Enter the student's score: 85
The student's grade is: H2
```

Modify the program to do the following:
(i) Insert a comment to say 'read score' in the appropriate location in the program to show where the score is input.

(ii) Modify the program to validate the score entered by the user. The score should be between 0 and 100 (inclusive). If the score is outside this range, the program should display an error message saying "Invalid score. Score should be between 0 and 100." and terminate without determining the grade.

When the program is run with an invalid score, the output may look as follows:

```
Enter the student's score: 150
Invalid score. Score should be between 0 and 100.
```

(iii) Extend the program to handle the case where the score is a floating-point number. The program should round the score to the nearest integer before determining the grade.

When the program is run with a floating-point score, the output may look as follows:

```
Enter the student's score: 78.6
The student's grade is: H3
```

(iv) Modify the program to determine the grade using a nested if-else structure instead of the elif statements.

(v) Incorporate the following function definition into your program and modify the program to call this function to determine the grade instead of using the if-else structure directly.

```
def determine_grade(score):
    if score ≥ 90:
        return 'H1'
    elif score ≥ 80:
        return 'H2'
    elif score ≥ 70:
        return 'H3'
    elif score ≥ 60:
        return 'H4'
    else:
        return 'H8'
```

When the program is run, the output should remain the same as before:

```
Enter the student's score: 92
The student's grade is: H1
```

(vi) Extend the program to display a message indicating whether the student passed or failed along with the grade. Additionally, one must add the missing grades-e.g. H5-H7.

When the program is run, the output may look as follows:

```
Enter the student's score: 75
The student's grade is: H3
```

```
The student passed.
```

Or:

```
Enter the student's score: 23
The student's grade is: H8
The student failed.
```

Save and close your file before moving on to the next part.

## Question 16(c):

Consider the following Python program that generates a random list of integers and performs various operations on the list:

```python
# Question 16(c)
# Examination Number:

import random

def generate_random_list(length, min_value, max_value):
    random_list = []
    for _ in range(length):
        random_list.append(random.randint(min_value, max_value))
    return random_list

# Generate a random list of 10 integers between 1 and 100
numbers = generate_random_list(10, 1, 100)
print("Original list:", numbers)
```

The program uses the `generate_random_list` function to create a list of 10 random integers between 1 and 100 and prints the original list.

Modify the program to do the following:
(i) Implement a function called `find_min_max` that takes a list of integers as input and returns a tuple containing the minimum and maximum values from the list. Use this function to find and print the minimum and maximum values from the `numbers` list.

(ii) Implement a function called `calculate_average` that takes a list of integers as input and returns the average of the numbers in the list. Use this function to calculate and print the average of the numbers in the `numbers` list, rounded to two decimal places.

(iii) Implement a function called `count_occurrences` that takes a list of integers and a target number as input and returns the count of occurrences of the target number in the list. Prompt the user to enter a target number and use the `count_occurrences` function to count and print the occurrences of the target number in the `numbers` list.

(iv) Implement a function called `remove_duplicates` that takes a list of integers as input and returns a new list with all duplicates removed, preserving the order of the elements. Use this function to remove duplicates from the `numbers` list and print the updated list.

(v) Implement a function called `find_second_largest` that takes a list of integers as input and returns the second largest number in the list. If the list has less than two unique elements, the function should return None. Use this function to find and print the second largest number in the `numbers` list.

(vi) Implement a function called `sort_ascending` that takes a list of integers as input and returns a new list with the elements sorted in ascending order. Use the bubble sort algorithm to implement the sorting. Use this function to sort the `numbers` list in ascending order and print the sorted list.

When the program is run, the output may look as follows:

```
Original list: [42, 15, 78, 92, 31, 56, 78, 22, 89, 61]
Minimum value: 15
Maximum value: 92
Average: 56.40
Enter a target number: 78
Count of occurrences of 78: 2
List with duplicates removed: [42, 15, 78, 92, 31, 56, 22, 89, 61]
Second largest number: 89
Sorted list: [15, 22, 31, 42, 56, 61, 78, 89, 92]
```

Note: The output may vary based on the randomly generated list.

Save and close your file before submitting your exam.

## Question 16(d):

Consider the following Python program that simulates a simple bank account management system:

```
# Question 16(d)
# Examination Number:
```

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance ≥ amount:
            self.balance -= amount
        else:
            print("Insufficient funds.")

    def display_balance(self):
        print(f"Account {self.account_number} balance: {self.balance}")


# Create bank accounts
account1 = BankAccount("001", 1000)
account2 = BankAccount("002", 500)

# Perform transactions
account1.deposit(500)
account2.withdraw(200)

# Display account balances
account1.display_balance()
account2.display_balance()
```

The program defines a `BankAccount` class with methods for depositing, withdrawing, and displaying the account balance. It creates two bank account objects, performs some transactions, and displays the account balances.

Modify the program to do the following:
(i) Implement a method called `transfer` within the `BankAccount` class that takes another `BankAccount` object and an amount as parameters. The method should transfer the specified amount from the current account to the other account. If the current account has insufficient funds, it should print an error message.

(ii) Create a `SavingsAccount` class that inherits from the `BankAccount` class. The `SavingsAccount` class should have an additional attribute called `interest_rate` and a method called `calculate_interest` that calculates and returns the interest earned based on the account balance and interest rate.

(iii) Implement a static method called `total_balance` within the `BankAccount` class that takes a list of `BankAccount` objects as input and returns the total balance of all the accounts.

(iv) Create a `Bank` class that has a list of `BankAccount` objects as an attribute. Implement methods within the `Bank` class to add an account, remove an account, and find an account by account number.

(v) Modify the `BankAccount` class to keep track of the transaction history. Each time a deposit, withdrawal, or transfer is made, the transaction should be recorded with a timestamp. Implement a method called `display_transactions` that prints the transaction history for an account.

(vi) Implement exception handling in the `withdraw` and `transfer` methods of the `BankAccount` class. If the withdrawal or transfer amount is greater than the account balance, raise a custom exception called `InsufficientFundsError` with an appropriate error message.

When the program is run, it should demonstrate the functionality of the newly implemented methods and classes. The output may look something like this:

```
Account 001 balance: 1500
Account 002 balance: 300
Transferring 200 from Account 001 to Account 002
Account 001 balance: 1300
Account 002 balance: 500
Creating a SavingsAccount with interest rate 0.05
SavingsAccount 003 balance: 1000
Interest earned on SavingsAccount 003: 50.0
Total balance of all accounts: 2800
Transaction history for Account 001:
2023-06-10 10:30:00 - Deposit: 500
2023-06-10 10:32:00 - Transfer to Account 002: 200
Attempting to withdraw 1500 from Account 002
Insufficient funds. Withdrawal amount exceeds account balance.
```

Note: The output may vary based on the specific implementation and test cases.

Save and close your file before submitting your exam.

## Question 16(e):

```
rainfall_data = [10.2, 5.8, 7.1, 12.5, 8.9, 6.4, 9.3, 11.7, 7.6, 8.2, 9.1,
```

```
6.8]
```

The program initializes a list called `rainfall_data` with 12 float values representing rainfall measurements.

Write a Python program to do the following:

**(i)** Calculate and display the average rainfall from the data.
When the program is run, the output may look as follows:

```
Average Rainfall: 8.63
```

**(ii)** Determine and display the number of rainfall measurements that are above the average.
When the program is run, the output may look as follows:

```
Measurements Above Average: 5
```

**(iii)** Implement a function that finds the longest sequence of consecutive measurements that are strictly increasing.
Name the function `find_longest_increasing_sequence` as shown in the function definition header:

```
def find_longest_increasing_sequence(data):
    # Function body goes here
```

The function accepts one parameter, `data`, which is the list of rainfall measurements. The function body should be coded to determine and return the length of the longest sequence of consecutive measurements that are strictly increasing.
An example of the function being used is shown:

```
print(find_longest_increasing_sequence(rainfall_data))
```

When the program is run, the output may look as follows:

```
Longest Increasing Sequence: 3
```

**(iv) Implement a function that determines the minimum number of measurements that need to be removed from the list to make the entire list strictly increasing.
Name the function `min_removals_for_increasing` as shown in the function definition header:

```
def min_removals_for_increasing(data):
    # Function body goes here
```

The function accepts one parameter, `data`, which is the list of rainfall measurements. The function body should be coded to determine and return the minimum number of measurements that need to be removed to make the entire list strictly increasing.
An example of the function being used is shown:

```
print(min_removals_for_increasing(rainfall_data))
```

When the program is run, the output may look as follows:

```
Minimum Removals for Increasing: 2
```

Save your file.