🕐

ST-2 Assignment:

Problem 1: Sum of Odd and Even Numbers in a 2D List:

Write a Python function that takes a 2D list as input and returns a tuple containing the sum of all even numbers, the sum of all odd numbers, and the count of each. The function should also handle cases where the matrix may be empty or contain non-integer values.

- **Sample Input**:

  `matrix = [[1, 2, 3], [4, 'a', 6], [7, 8, 9]]`

- **Sample Output**:

  `(Even sum: 20, Odd sum: 25, Even count: 3, Odd count: 5)`

Problem 2: Advanced Notepad Implementation:

Write a Python program that simulates a basic Notepad application with the following functionalities:

- **Find**: Searches for a specified word or phrase in a given text.
- **Replace**: Replaces all occurrences of a specified word with another word.
- **Count Occurrences**: Counts how many times the specified word appears in the text.
- **Reverse Text**: Reverses the entire text string.

Implement these functionalities in separate functions within the program. Handle case sensitivity and return appropriate messages when a word is not found.

- **Sample Input**:

  `text = "Python is fun. Python is easy."`

  Find: `"Python"`

  Replace with: `"JavaScript"`

- **Sample Output**:

```
"JavaScript is fun. JavaScript is easy."
Occurrences of 'Python': 2
Reversed Text: ".ysae si nohtyP .nuf si nohtyP"
```

Problem 3: Student Management System

Write a Python program that implements a simple student management system using dictionaries. The program should support the following functionalities:

1. **Add a Student**: Allow the user to input a student's name, age, and grades. If the student already exists, inform the user.
2. **Remove a Student**: Allow the user to remove a student by their name.
3. **Update Student Details**: Allow the user to update a student's age or grades.
4. **Display All Students**: Show the details of all students currently in the system.
5. **Calculate Average Grades**: Calculate and display the average grade of all students.

Input Specifications:

- The program should accept commands from the user in the following format:
  - **Add Student**: `add <name> <age> <grade1> <grade2> ... <gradeN>`
  - **Remove Student**: `remove <name>`
  - **Update Student**: `update <name> age <new_age>` or `update <name> grades <new_grade1> <new_grade2> ... <new_gradeN>`
  - **Display Students**: `display`
  - **Calculate Average**: `average`
  - **Exit**: `exit`

Output Specifications:

- For each command:
  - When adding a student, print `"Student '<name>' added successfully."` or `"Student '<name>' already exists."`

- When removing a student, print `"Student '<name>' removed successfully."` or `"Student '<name>' not found."`
- When updating, print `"Student '<name>' updated successfully."` or `"Student '<name>' not found."`
- When displaying, show all students in the format: `Name: <name>, Age: <age>, Grades: <grades>`
- When calculating average, display: `Average grade of all students: <average>`

Sample Input/Output:

```
Input:
add Alice 20 85 90 88
add Bob 22 78 82 80
display
update Alice age 21
update Bob grades 80 85 90
average
remove Bob
display
exit

Output:
Student 'Alice' added successfully.
Student 'Bob' added successfully.
Name: Alice, Age: 20, Grades: [85, 90, 88]
Student 'Alice' updated successfully.
Student 'Bob' updated successfully.
Average grade of all students: 84.25
Student 'Bob' removed successfully.
Name: Alice, Age: 21, Grades: [85, 90, 88]
```

Explanation:

- **Functionality**: The program allows users to manage student data through a command interface, providing a practical application of dictionary manipulation.
- **Sample Interaction**: The provided sample input/output shows how users would interact with the program, making it easy for students to understand

the expected behavior.

Problem 4: Sum of Unique Elements in a 2D List

Problem Statement:
Imagine you are managing a small bakery, and you want to keep track of the unique ingredients used in each recipe. Each recipe is represented by a row in a 2D list, where the elements in the row signify the ingredients used (some ingredients may repeat). Your task is to write a Python function that computes the sum of all unique ingredient quantities for each recipe.

Help the bakery ensure they only count each unique ingredient once in their inventory list!

Sample Input:

```
matrix = [[1, 2, 2], [3, 4, 3], [5, 5, 6]]
```

Sample Output:

```
Sum of unique elements in each row: [3, 7, 11]
```

Problem 5: Problem Statement:

You are creating a text analysis tool that needs to analyze various phrases and their character frequencies. Given a tuple containing multiple phrases, your challenge is to write a Python program that counts the frequency of each character across all phrases, regardless of the phrase they belong to. The output should be a dictionary showing each character and its corresponding frequency, providing valuable insights into the text.

Help unlock the mysteries of language by revealing the frequency of each character used in the provided phrases!

Sample Input:

```
strings = ("hello", "world", "hello world")
```

Sample Output:

```
Character frequencies: {'h': 2, 'e': 2, 'l': 5, 'o': 3,
```

```
'w': 2, 'r': 1, 'd': 1}
```

Problem 6: Employee Management System

You are tasked with developing a simple Employee Management System for a small company. The company keeps track of its employees' details, including their names, ages, departments, and salaries.

Below is a list of employees represented as dictionaries:

```
employees = [
    {"name": "Alice", "age": 30, "department": "HR", "salar
y": 60000},
    {"name": "Bob", "age": 25, "department": "Engineering",
"salary": 75000},
    {"name": "Charlie", "age": 35, "department": "Finance",
"salary": 80000},
    {"name": "David", "age": 40, "department": "HR", "salar
y": 90000},
    {"name": "Eve", "age": 28, "department": "Engineering",
"salary": 70000},
]
```

Tasks

You are required to implement the following functionalities:

1. **Filter Employees by Department**: Create a function that accepts a department name as input and returns a list of employee names who belong to that department. Implement this using a higher-order function.
2. **Calculate Average Salary**: Develop a function that calculates the average salary of all employees. Use a list comprehension to compute this value.
3. **Find Employee by Name**: Implement a function that takes an employee's name as input and returns their details (age, department, and salary) as a dictionary. If the employee is not found, return `None`. Use a list comprehension to facilitate the search.
4. **Count Employees by Age Group**: Write a function that categorizes employees into age groups: under 30, 30-40, and over 40. Return a

dictionary containing the counts for each age group. Utilize a combination of `map` and `sum` to achieve this.

5. **Sort Employees by Salary**: Create a function that sorts the employees by their salary in descending order and returns a list of employees in that order. Use the `sorted()` function with a lambda as the key for sorting.

Expected Outcomes

After implementing the above functionalities, you should be able to perform various operations on the employee data efficiently. The expected outputs for the functions should match the following examples:

- For the department filter function with input `"Engineering"`, the output should be: `["Bob", "Eve"]`.
- The average salary function should return: `75000.0`.
- The find employee function for `"Charlie"` should return: `{"name": "Charlie", "age": 35, "department": "Finance", "salary": 80000}`.
- The count by age group function should output: `{"under_30": 2, "30_to_40": 3, "over_40": 1}`.
- The sorted employees function should return:

```
[
    {"name": "David", "age": 40, "department": "HR", "sa
lary": 90000},
    {"name": "Charlie", "age": 35, "department": "Financ
e", "salary": 80000},
    {"name": "Bob", "age": 25, "department": "Engineerin
g", "salary": 75000},
    {"name": "Eve", "age": 28, "department": "Engineerin
g", "salary": 70000},
    {"name": "Alice", "age": 30, "department": "HR", "sa
lary": 60000},
]
```

Problem 7: Diagonal Sums of a 2D List

Write a Python function that takes a square 2D list (matrix) as input and calculates the sums of both the left and right diagonals. The left diagonal consists of elements where the row index and column index are the same (e.g., matrix[0][0], matrix[1][1]), while the right diagonal consists of elements where the row index and column index sum to the length of the matrix minus one (e.g., matrix[0][n-1], matrix[1][n-2]).

- **Sample Input:**

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

- **Sample Output:**

```
Left Diagonal Sum: 15, Right Diagonal Sum: 15
```

Problem 8: Common Elements Between Tuple and Set

Write a Python function that accepts a tuple and a set as input and returns a new set containing only the elements that are common to both the tuple and the set. Make sure to use set operations to achieve this.

- **Sample Input:**

```
my_tuple = (1, 2, 3, 4, 5)
my_set = {3, 4, 5, 6, 7}
```

- **Sample Output:**

```
Common Elements: {3, 4, 5}
```

Problem 9: Employee Records Analysis

You are given a list of employee records where each record is represented as a dictionary. Each dictionary contains the following keys: `"name"`, `"age"`,

`"department"`, and `"salary"`. Your task is to implement a function that performs the following analyses:

1. **Unique Departments:** Create a set of all unique departments represented in the records.
2. **Average Salary:** Calculate the average salary of employees in each department and return it as a dictionary where the keys are department names and the values are the average salaries.
3. **Highest Paid Employee:** Identify the highest-paid employee and return a tuple containing the employee's name and their salary.
4. **Employees Over Age:** Return a list of names of employees who are older than a specified age.

- **Sample Input:**

```
employees = [
    {"name": "Alice", "age": 30, "department": "HR", "salar
y": 70000},
    {"name": "Bob", "age": 24, "department": "IT", "salar
y": 60000},
    {"name": "Charlie", "age": 28, "department": "HR", "sal
ary": 80000},
    {"name": "David", "age": 35, "department": "IT", "salar
y": 90000},
    {"name": "Eve", "age": 29, "department": "Finance", "sa
lary": 75000}
]
age_limit = 27
```

- **Sample Output:**

```
Unique Departments: {'Finance', 'HR', 'IT'}
Average Salaries: {'HR': 75000.0, 'IT': 75000.0, 'Finance': 7
Highest Paid Employee: ('David', 90000)
Employees Over Age: ['Alice', 'Charlie', 'David', 'Eve']
```