1. **CSV Column Averager**: Write a function that reads a CSV file and calculates the average of the numbers in a given column.

Hint: Use the **csv** module to read the file and remember to skip the header row. Convert the column values to floats before averaging.

1. **Anagram Checker**: Create a function that checks whether two strings are anagrams of each other.

Hint: Use a dictionary to count the occurrences of each character in both strings and then compare.

1. **Recursive Factorial**: Write a recursive function that returns the factorial of a given number.

Hint: Remember that the factorial of a number is the product of all positive integers up to that number.

1. **Nested Dictionary Updater**: Develop a function that takes a nested dictionary and updates a value given its key path as a list.

Hint: Use recursion or iteration to navigate the nested dictionaries.

1. **Custom Sorter**: Write a function that sorts a list of tuples based on the second value in each tuple.

Hint: Use the **sorted** function with a custom **key** parameter.

1. **Leap Year Verifier**: Create a function that checks if a given year is a leap year.

Hint: Use the rules that define leap years (divisible by 4, not by 100 unless also by 400).

1. **Palindromic Substrings**: Write a function that finds and returns all the palindromic substrings of a given string.

Hint: Consider using a sliding window technique to generate all possible substrings.

1. **Two-Sum Problem**: Develop a function that takes a list of numbers and a target sum, returning the indices of the two numbers that add up to the target sum.

Hint: Consider using a dictionary to store numbers and their indices for quick lookups.

1. **Matrix Transposer**: Write a function that transposes a matrix represented as a list of lists.

Hint: Nested list comprehensions can be used to invert rows and columns.

1. **Custom String Formatter**: Create a function that takes a string and capitalizes every other letter in the string.

Hint: Use slicing with step and string concatenation or a **join** operation.

1. **Max Heap Checker**: Write a function that checks if a given list of numbers represents a valid max heap.

Hint: For each element at index **i**, ensure that **heap[i] >= heap[2\*i+1]** and **heap[i] >= heap[2\*i+2]** if those indices exist.

1. **Circular Prime Checker**: Develop a function that checks if all rotations of a number's digits are prime.

Hint: Generate all rotations of the number and check each one for primality.

1. **HTML Tag Validator**: Write a function that validates a string of HTML, ensuring all tags are properly opened and closed in the correct order.

Hint: Use a stack to keep track of opening tags and match them with closing tags.

1. **Longest Consecutive Sequence**: Create a function that finds the length of the longest consecutive elements sequence in an unsorted list.

Hint: Use a set to achieve O(n) complexity, eliminating the need to sort the list.

1. **Regular Expression Matcher**: Write a function that uses regular expressions to check if a given string matches a given pattern.

Hint: The **re** module in Python provides pattern matching utilities.