
Assignment 14 – Managerial Assignment (TMA)

Course: Software Design and Development in Python (20606)
Assignment Topics: Efficiency, Recursion
Learning Material for the Assignment: Units 8-10
Assignment Weight: 2 Credits
Number of Questions: 4
Submission Deadline: 17.5.2025
Semester: 2025b

Please Note:

- You **must strictly adhere** to the assignment instructions as written.
- Follow the **PEP 8 standard** as demonstrated in **Unit 1.9** on the course website.
- **Do not include explanations** beyond those requested in the assignment instructions unless explicitly asked. If written, explanations **must be in English**.
- **Do not use advanced material or concepts not taught in the course.** Specifically, do not use data structures like **dictionaries (`dict`)**, nor implement **classes and programs independently**.
- **Do not use global variables or global constants at all.**
- **Comments** should appear where necessary and **be written in English**.
- Answers should be written **accurately and clearly** according to the question's requirements. **Proper indentation, clear writing, and appropriate spacing will receive significant weight in grading.**
- **Submit only one file named exactly as follows:** `mmn14.py`. Save all the solutions for the assignment in this file, **clearly separated by comments**. If you submit multiple files or incorrectly named files, **your assignment may not be checked**.
- **Do not forget to clearly indicate the question numbers in the code comments.**.

Question 1 – (25%)

The K-Shift Rotation:
In a K-shift rotation, **each element of the list is moved K positions forward (right)** in the list, **wrapping around if necessary**. If the shift moves beyond the last position, **the rotation continues from the start of the list**.

For example, for the following list:

```
lst = [2, 6, 11, 17, 40, 42]
```

After performing a **K-shift rotation with K = 3**, the list becomes:

```
lst = [17, 40, 42, 2, 6, 11]
```

Write a function named **`find_max`** that:

- **Receives a list `lst`.**
- **Iterates through its numbers in ascending order starting from the position corresponding to K.**
- Assume **K is a non-negative integer between 0 and the length of the list (inclusive)**.
- The function should **return the maximum value found** during the iteration.
- **If the list is empty, return `None`.**

Important Notes:

- Your solution must be **efficient in both time and space complexity**.
- A solution with **higher time complexity will receive only minor partial credit or even a score of zero for this question.**
- **Do not create any additional data structures (including new lists).**
- **Do not sort the list `lst` at any stage.**
- **The original list `lst` must remain unchanged.**

📌 **Additional Comments:**

- **You can assume the parameter `lst` is always a list of numerical values.**
- **You can assume all values in the list are unique.**
- The function must **return only the maximum value found starting from position K** (following the rotation logic) and **must not return a modified list or any additional data.**

Question 2 – (25%)

Write a function named **`find_pairs`** that:

- **Receives a sorted list `lst` (ascending order, no duplicates).**
- **Receives a number `k`.**
- **Returns the number of pairs whose difference is exactly `k`.**
- The pairs **do not have to be adjacent in the list**.
- **If no such pairs exist, return 0.**

📌 **Important:** The function should calculate **differences between values**.

Example:

For the list:

```
lst = [-7, -3, 0, 1, 3, 5, 12, 14, 17, 19, 25, 30]
```

- For **`k = 2`**, return **4 pairs**:
  `(1, -1), (3, 1), (5, 3), (17, 15)`
- For **`k = 4`**, return **2 pairs**:
  `(-3, -7), (1, -3)`
- For **`k = 6`**, return **2 pairs**:
  `(19, 13), (25, 19)`
- For **`k = 23`**, return **0 pairs** (no such difference found).

✅ **Strict Requirements:**

- Your solution **must be as efficient as possible**.
- **Solutions with poor efficiency will not receive credit.**
- You **must perform a single pass over the list (`lst`) only.**
- **Do not use any additional data structures (including new lists, sets, or dictionaries).**
- **You must not sort the list (it is already sorted).**
- **The original list `lst` must remain unchanged.**

📌 **Additional Comments:**

- **You can assume the parameter `lst` is always a sorted list of numerical values.**
- **The function must process `lst` directly, without copying or creating any additional structures.**

Thanks for the clarification! Here's the **final and fully aligned version with all key constraints and requirements clearly highlighted**:

⚠️ Attention!

In **Questions 3-4**, it is **forbidden to use global variables or constants of any kind!**

Question 3 – (25%)

Part A

Write a **recursive function** named `update_list` that:

- **Receives a list `lst` of numbers and a value `value`.**
- The function **updates the list by removing the first occurrence of `value` from the list, if it exists**.
- If the value **does not exist in the list, the function returns the list unchanged**.
- If the value **appears more than once, only the first occurrence should be removed**.

Example:

```
lst = [3, 8, 10, 6]
update_list(lst, 1)  ➔  [3, 8, 10, 6]    # **value 1 not found**

lst = [4, 1, 3]
update_list(lst, 3)  ➔  [4, 1]           # **value 3 removed**

lst = [3, 8, 10, 6]
update_list(lst, 8)  ➔  [3, 10, 6]       # **value 8 removed**
```

📝 **Notes:**

- **You may write helper functions and/or use parameter overloading with default values.**
- **You must NOT use the `in` operator or the `remove` method.**
- **Loops are also forbidden.**

Part B

Write a **recursive function** named **`equal_lists`** that:

- **Receives two lists, `lst1` and `lst2`, both containing numbers.**
- Returns **`True` if both lists contain exactly the same elements, (in ANY ORDER), and the same quantity of each element.**
- Otherwise, returns **`False`**.

Example:

```
lst1 = [4, 3, 1, 2]
lst2 = [1, 2, 3, 4]
equal_lists(lst1, lst2)  ➔  **True**

lst1 = [8, 1, 3]
lst2 = [8, 1]
equal_lists(lst1, lst2)  ➔  **False**
```

📝 **Notes:**

- **You may use the `update_list` function you wrote in Part A.**
- **You must NOT use the `in` operator, the `remove` method, or any loops.**

Question 4 – (25%)

A **palindrome** is a word, number, sentence, or sequence that reads the same backward as it does forward.

A **palindromic list** is a list consisting of palindromes, where the list itself is also a palindrome (i.e., the list reads the same backward as forward, and all its elements are palindromes).

Examples:

- The list `lst1 = ['abba', 'readaer', 'abba']` is a **palindromic list** because:
  1. The list itself is a palindrome.
  2. All its elements are also palindromes.
- The list `lst2 = ['abba']` is also a **palindromic list**.
- The list `lst3 = ['a', 'aa', 'aba', 'a']` is **not a palindromic list**.
  While all its elements are palindromes, the list itself is **not** a palindrome.
- The list `lst4 = ['abba', 'gonna', 'abba']` is **not a palindromic list**.
  Although the list itself is a palindrome, the element `'gonna'` is **not** a palindrome.

Write a function named **`is_palindrome`** that:

- **Receives a list `lst` of strings.**
- **Returns `True` if the list is a palindromic list; otherwise, returns `False`.**

📝 **Notes:**

- **An empty list or a list containing only empty strings is considered a palindromic list.**
- **You may write helper functions and/or use recursion with additional parameters if needed.**
- **Do not use loops or string methods like slicing (e.g., `[::-1]`).**
- **The function must access the list `lst` using indexing only (no loops or comprehensions).**
