Laboratory Exercise 5

Topics

1. Lists of items
2. Use the for loop to scroll through a list
3. Functions that use lists

Discussion

A. What is a list?
B. What are the differences between an element of a list, and its index?
C. What list's information is needed to scroll it without making mistakes?

Exercises

Part 1 – Elaboration of lists

5.1.1 Sum with alternating signs. Write a program that receives as input a sequence of integers (terminated by an empty line), and that calculates the alternating sum of its elements. For example, if the program reads the data `1 4 9 16 9 7 4 9 11`, it must calculate and display `1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11 = -2`. [P6.8]

5.1.2 List of random numbers. Write a program that initializes a list with ten random integers between `1` and `100` and then displays, on four successive lines:

I. `All elements of even index;`
II. `All the elements that are even;`
III. `All elements in reverse order;`
IV. `The first and the last element.`

[P6.1]

5.1.3 Remove the minimum value. Write a `remove_min(v)` function that removes the minimum value from a `v` list without using the `min()` function or the `remove()` method. [P6.7]

5.1.4 Local highs. Read a sequence of integers ended by a blank line. Print the position of the local maxima (numbers greater than both the previous and the next value) if there are any, otherwise print

the message `'There are no local maxima'`. Extension: if there are several pairs of local maxima, identify the two closest local maxima and print their position.

5.1.5 The same elements. Write the `same_set(a, b)` function that checks if two lists contain the same elements, regardless of the order and ignoring the presence of duplicates. For example, the two lists `1 4 9 16 9 7 4 9 11` and `11 11 7 9 16 4 1` must be considered equal. The function must not modify the lists that have been passed as parameters. [P6.12]

5.1.6 Ordered list. Write a program that generates a sequence of `20` random integer values between `0` and `99`, then displays the generated sequence, orders it, and displays it again, sorted. Use the `sort()` method. [P6.17]

5.1.7 Add up without the minimum. Write the `sum_without_smallest(v)` function that calculates the sum of all the values of a list `v`, excluding the minimum value. [P6.6]

Part 2 – Algorithms that make use of lists

5.2.1 Measurement noise. Often the data collected during an experiment must be processed to remove some of the measurement noise. A simple approach to this problem involves replacing, in a list of values, each value with the average between the same value and the two adjacent values (or of the only adjacent value if the value under consideration is at one of the two ends of the list). Write a program that does this, without creating a second list. [P6.36]

5.2.2 Distances. People who park their car in a row of parking lots usually prefer to maximize the distance between the seat they occupy and the seats that are already occupied by other vehicles. They therefore tend to occupy the central seat of the longest row of free seats available. For example, consider the situation where ten seats are free:

```
_ _ _ _ _ _ _ _ _ _
```

The first person to arrive will occupy, with their vehicle, a seat in the central part of the row:

```
_ _ _ _ _ X _ _ _ _
```

The next person will place it in the middle of the longest vacated row (i.e. the one on the left):

```
_ _ X _ _ X _ _ _ _
```

Write a program that receives as input the number of parking spaces that make up the row of parking spaces and that, each time a new space is occupied according to the rule indicated, displays the row in the format indicated above. Tip: Use a list of Boolean values to indicate whether a parking space is occupied or not. [P6.19]

5.2.3 Bulgarian solitaire. The Bulgarian Solitaire game starts with `45` cards. The cards are randomly divided into a given number of randomly sized piles. For example, you can start with 5 piles of sizes `20, 5, 1, 9 and 10`. Each turn, one card is subtracted from each pile, and a new pile is created made up of the cards subtracted from the starting piles. For example, the starting configuration would be transformed into stacks of sizes `19, 4, 8, 9 and 5`. The game ends when the stacks have sizes `1, 2, 3, 4, 5, 6, 7, 8 and 9` (you can show that you always get such a configuration). Write a program that generates a random initial setup and displays it, and then continues with the solitaire steps, displaying the setup after each step, and stopping when you reach the final solitaire setup. [P6.20]