Laboratory 09

Topics

1. Using conditional constructs for decision making (Lab03 recap)
2. Using cycles to execute repeating instructions (Lab04 recap)
3. Definition and elaboration of lists and tables (Lab08 recap)

Discussion

A. How are conditional constructs used for complex decision making?
B. A complex decision can be based on general and/or more specific conditions. How should the conditional construct manage them?
C. In cyclic programming, what is an `off-by-one error`?
D. What is the search algorithm used to find an element in a set of unsorted elements?
E. Which constructor is more suitable to access the elements of a table with rows and columns?

Exercises

Part 1 – Recap Exercises

Task: *For each of the following exercises, write a program in Python that meets the given requirements. Complete at least one exercise during the laboratory session, and the remaining ones at home.*

09.1.1 Lista of functions. Write list functions that carry out the following tasks for a list of integers. For each function, provide a test program. All the operations must modify the list passed as a parameter. [P6.4]

I. Swap the first and last elements in the list.
II. Shift all elements by one to the right and move the last element into the first position. For example, `1 4 9 16 25` would be transformed in `25 1 4 9 16`.
III. Replace all even elements with `0`.
IV. Replace each element except the first and last by the larger of its two neighbors.
V. Remove the middle element if the list length is odd, or the middle two elements if the length is even.
VI. Remove the middle element if the list length is odd, or the middle two elements if the length is even.
VII. Return the second-largest element in the list.
VIII. Return `True` if the list is currently sorted in increasing order.
IX. Return `True` if the list contains two adjacent duplicate elements.
X. Return `True` if the list contains duplicate elements (which need not be adjacent).

09.1.2 Hidden rules. Write a program that, given an empty list `l = []`, initializes it with each of the following sequences, after you have understood the rules used to generate them (if they exist): [R6.1]

I. `1 2 3 4 5 6 7 8 9 10`
II. `0 2 4 6 8 10 12 14 16 18 20`

III. `1 4 9 16 25 36 49 64 81 100`
IV. `0 0 0 0 0 0 0 0 0 0`
V. `1 4 9 16 9 7 4 9 11`
VI. `0 1 0 1 0 1 0 1 0 1`
VII. `0 1 2 3 4 0 1 2 3 4`

09.1.3 Bar chart. Write a program that reads a sequence of input values and displays a bar chart of the values, using asterisks (`*`), like this:

```
**********
****************************************
****************************
*************************
**************
```

You may assume that all values are positive. First figure out the maximum value. That value's bar should be drawn with `40` asterisks. Shorter bars should use proportionally fewer asterisks. [P6.24]

09.1.4 What about negative numbers? Improve the program of **Exercise 09.1.3** to work correctly when the data set contains negative values. [P6.25]

09.1.5 Caption. Improve the program of **Exercise 09.1.3** by adding captions for each bar. Prompt the user for the captions and data values. The output should look like this:

```
      Egypt **********
     France ****************************************
      Japan ****************************
    Uruguay *************************
Switzerland **************
```

[P6.26]

09.1.6 List of integer numbers. Write a program that reads from input a list of positive integer numbers provided on single row and separated by the character `':'`, for example : `3:12:21:8:4:7`. The program must print, maintaining the same format:

I. The input numbers, excluding the maximum and the minimum (for example, `12:8:4:7`);
II. Among the inputted numbers, only the even values (for example, `12:8:4`);
III. Among the inputted numbers, only the 2-digits numbers (for example, `12:21`).

Part 2 – Applications

Task: *For each of the following exercises, write a program in Python that meets the given requirements. Complete at least one exercise during the laboratory session, and the remaining ones at home.*

09.2.1 The theater. A theater seating chart is implemented as a table of ticket prices, like this:

```
10, 10, 10, 10, 10, 10, 10, 10, 10, 10
10, 10, 10, 10, 10, 10, 10, 10, 10, 10
10, 10, 10, 10, 10, 10, 10, 10, 10, 10
10, 10, 20, 20, 20, 20, 20, 20, 10, 10
10, 10, 20, 20, 20, 20, 20, 20, 10, 10
10, 10, 20, 20, 20, 20, 20, 20, 10, 10
20, 20, 30, 30, 40, 40, 30, 30, 20, 20
20, 30, 30, 40, 50, 50, 40, 30, 30, 20
30, 40, 50, 50, 50, 50, 50, 50, 40, 30
```

Write a program that prompts users to pick either a seat or a price. Mark sold seats by changing the price to `0`. When a user specifies a seat, make sure it is available. When a user specifies a price, find any seat with that price. [P6.27]

09.2.2 Concatenated words. While traveling, a good way to pass time is to play the game "concatenated words". The first player starts by telling an initial word, then, in turns, the following player should tell a word that starts with last `2` letters of the word said by the previous player. It is forbidden to say a word that has been said in a previous turn. An example is the sequence: `'cat', 'attic', 'ice', 'cerulean', 'animal'`. Write a program that allows to manage one or more games. Each game ends when one of the players enters:

1. a word that has already been said in a previous turn;
2. an invalid word;
3. an asterisk (`*`) to quit the game. This character can also be inputted when the player is unable to find a valid word.

09.2.3 The best client. A supermarket wants to reward its best customer of each day, showing the customer's name on a screen in the supermarket. For that purpose, the customer's purchase amount is stored in a list (`customers`) and the customer's name is stored in a corresponding list (`customers`). Implement a function:

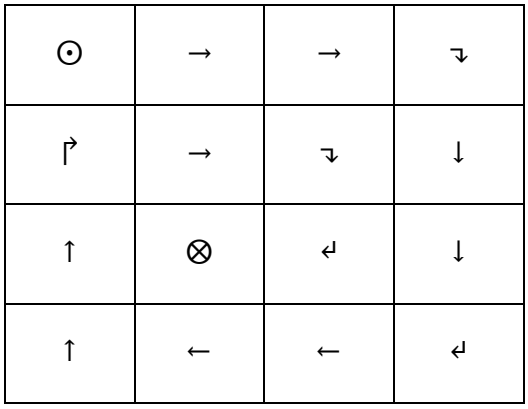
```
name_of_best_customer(sales, customers)
```

that returns the **name** of the customer with the largest sale.

One you have done that, write a program that prompts the cashier to enter all prices and names, adds them to two lists, calls the function that you implemented, and displays the result. Use a price of `0` as a sentinel. [P6.33]

09.2.4 Spiral. Write a program that asks the user to input a value `N`, builds a matrix `N x N`, fills it with a sequence of integer numbers between `1` and `N**2`, and prints the resulting table. For example, if `N = 4`, the program should print the table on the left, following the instructions shown in the table on the right (this table should not be an output of your program).

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7



09.2.5 Pet shop. A pet shop wants to give a discount to its clients if they buy one or more pets and at least five other items. The discount is equal to 20 percent of the cost of the other items, but not the pets. Implement a function:

```
discount(prices, is_pet)
```

The function receives information about a particular sale and computes the discount to apply. For the i-th item, `prices[i]` is the price before any discount, and `is_pet[i]` is a Boolean variable that is `True`if the item is a pet.

Once you have done that, write a program that prompts a cashier to enter each price and then a `'Y'` for a pet or `'N'` for another item. Use a price of `-1` as a sentinel. Save the inputs in a list. Call the function that you implemented and display the discount. [P6.32]

09.2.6 Flooding. You are given a $N \times N$ table `heights` of real values that give the height of a terrain at different points in a square. Write a function:

```
flood_map(heights, water_level)
```

that prints out a flood map, showing which of the points in the terrain would be flooded if the water level reaches the value `water_level`. The flood map is an $N \times N$ table of strings, each composed by a single character. In the flood map, print a `'*'` for each flooded point and a space `' '` for each point that is not flooded. Here is a sample map:

```
* * * *         * *
* * * * *     * * *
* * * *         * *
* * *       * * * *
* * * *     * * * *
* * * * * * * * * *
* * * *       * * *
* *    * *     * *
              * *
```

Once you have done that, write a program that builds a `10x10` height matrix, reading one hundred terrain height values and shows how the terrain gets flooded when the water level increases in `10` steps from the lowest point in the terrain to the highest [P6.35]