# Esercitazione di Laboratorio 08

## Temi trattati

1. Liste
2. Tabelle

## Discussione

A. Discutere somiglianze e differenze tra liste e tabelle.
B. Descrivere l'uso degli indici in:
    a. stringhe;
    b. liste;
    c. tabelle.
C. Come si previene un `out-of-range error`?

## Esercizi

### Parte 1 – Elaborazione di liste e tabelle

***Consegna**: per ciascuno degli esercizi seguenti, scrivere un programma in Python che risponda alle richieste indicate. Completare almeno due esercizi durante l'esercitazione, e i rimanenti a casa.*

**08.1.1 Out-of-range.** Scrivere un programma che contenga un `out-of-range error`. Cosa succede se si esegue il programma? [R6.10]

**08.1.2 Buffer.** Scrivere lo pseudocodice per un algoritmo che, data una lista di lunghezza definita, sposta gli elementi "in avanti" di una posizione (incrementando quindi il loro indice di una unità), e sposta l'elemento all'ultima posizione nella prima posizione. Ad esempio, la lista `1 7 9 3 0 4`, dopo questa operazione, diventa la lista `4 1 7 9 3 0`. Scrivere poi un programma che implementi l'algoritmo così definito. [R6.17]

**08.1.3 Giocare a dadi.** Scrivere un programma che generi una sequenza di `20` lanci casuali di dadi, li memorizzi in una lista e visualizzi i valori generati, contrassegnando la serie di valori identici più lunga con questa formattazione:

```
1 2 5 5 3 1 2 4 3 (2 2 2 2) 3 6 5 5 6 3 1
```

Se sono presenti più sequenze di valori identici di lunghezza massima, si usi la formattazione indicata per evidenziare la prima in ordine di occorrenza. [P6.16]

**08.1.4 Tabella.** Scrivere le istruzioni Python per eseguire le seguenti operazioni con una tabella di $m$ righe e $n$ colonne (dimensioni inserite da tastiera):

I. inizializzare la tabella con valori pari a zero (`0`);
II. riempire tutte le caselle con valori pari a uno (`1`);
III. riempire le caselle alternando `0` e `1` in uno schema a scacchiera;
IV. riempire di `0` solo le caselle della riga superiore e di quella inferiore, lasciando invariato il resto della tabella;
V. riempire con `1` solo le caselle della colonna di destra e di sinistra, lasciando invariato il resto della tabella;

VI. calcolare e stampare la somma di tutti gli elementi.

Dopo ogni operazione, stampare la tabella. [R6.28]

**08.1.5 Unione di liste.** Scrivere una funzione `merge(a, b)` che unisca le due liste `a` e `b`, alternando un elemento della prima e un elemento della seconda. Se una lista è più corta dell'altra, gli elementi vengono alternati fin quando è possibile, poi gli elementi rimasti nella lista più lunga vengono aggiunti, in ordine, in fondo. Le liste di partenza non devono essere modificate. Se, ad esempio, il contenuto di `a` è `1 4 9 16` e il contenuto di `b` è `9 7 4 9 11`, l'invocazione di `merge(a, b)` restituisce una nuova lista contenente i valori `1 9 4 7 9 4 16 9 11`. [P6.30]

**08.1.6 Unione di liste ordinate.** Scrivere la funzione `merge_sorted(a, b)` che unisca due liste (che si assumono già ordinate in modo crescente), restituendo una nuova lista ordinata in modo crescente. Gestire un indice corrente per ciascuna lista, in modo da tenere traccia delle porzioni già elaborate. Le liste di partenza non devono essere modificate. Se, ad esempio, il contenuto di `a` è `1 4 9 16` e il contenuto di `b` è `4 7 9 9 11`, l'invocazione `merge_sorted(a, b)` restituisce una nuova lista contenente i valori `1 4 4 7 9 9 9 11 16`. Non utilizzare il metodo `sort()` né la funzione `sorted()`. [P6.31]

## Parte 2 – Algoritmi che fanno uso di liste e tabelle

***Consegna**: per ciascuno degli esercizi seguenti, scrivere un programma in Python che risponda alle richieste indicate. Completare almeno un esercizio durante l'esercitazione, e i rimanenti a casa.*

**08.2.1 Mediamente.** Scrivere la funzione `neighbor_average(values, row, column)` che, in una tabella `values`, calcoli il valore medio dei vicini di un elemento nelle otto direzioni, indicizzati come illustrato nella figura sotto (escludendo l'elemento stesso). Se, però, l'elemento si trova su un bordo della tabella, la media va calcolata considerando soltanto i vicini che appartengono effettivamente alla tabella. Ad esempio, se `row` e `column` valgono entrambe `0`, l'elemento ha `3` vicini. [P6.23]

| | | |
|---|---|---|
| `[i -1] [j - 1]` | `[i - 1] [j]` | `[i - 1] [j + 1]` |
| `[i] [j - 1]` | `[i] [j]` | `[i] [j + 1]` |
| `[i + 1] [j - 1]` | `[i + 1] [j]` | `[i + 1] [j + 1]` |

**08.2.2 Quadrati magici.** Una matrice $n \times n$ contenente i numeri interi $1, 2, 3, \ldots, n^2$ è un "quadrato magico" se la somma dei suoi elementi in ciascuna riga, in ciascuna colonna e nelle due diagonali ha lo stesso valore. Ad esempio, questo è un quadrato magico di dimensione 4:

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

Scrivere un programma che acquisisca in ingresso 16 valori, li disponga in una tabella 4 × 4 in ordine, una riga alla volta dall'alto in basso, e in ciascuna riga da sinistra a destra, e verifichi se, dopo averli disposti, questi formano un quadrato magico. Verificare due proprietà:

I. Nei dati acquisiti sono presenti tutti e solamente i numeri 1, 2, ..., 16.
II. Quando i numeri vengono disposti nella tabella, le somme delle righe, delle colonne e delle diagonali sono tutte uguali l'una all'altra. [P6.21]

**08.2.3 Il gioco del tris.** Scrivere un programma che giochi a tris. Il gioco del tris si svolge su una griglia 3 × 3. Il gioco è giocato da due giocatori umani che si alternano. Il primo giocatore segna le mosse con un cerchio ('o'), il secondo con una croce ('x'). Vince il giocatore che ha formato una sequenza orizzontale, verticale o diagonale di 3 segni. Il programma deve, ad ogni turno, visualizzare il tabellone di gioco, chiedere in input all'utente le coordinate del prossimo segno (riga e colonna, numerate da 1 a 3), invertire i giocatori dopo ogni mossa e, finita la partita, decretare il vincitore o la condizione di parità. [P6.28]

| | | |
|---|---|---|
| X | | O |
| | X | O |
| O | O | X |

**08.2.4 Molla.** Scrivere un programma che modelli e simuli il movimento di un oggetto di massa $m$ collegato ad una molla oscillante. Quando la molla viene spostata dalla sua posizione di equilibrio di una quantità $x$ , la legge di Hooke afferma che la forza di ripristino è data dalla formula:

$$F = -kx$$

dove $k$ è una costante che dipende dalla molla. Per questa simulazione, utilizzare `k = 10 N/m`. Iniziare con un determinato spostamento $x$ (ad esempio, `x = 0.5 m`). Impostare la velocità iniziale `v = 0`. Calcolare l'accelerazione $a$ in base alla legge di Newton ($F = ma$ ) e alla legge di Hooke, utilizzando una massa `m = 1 kg`. Utilizzare un piccolo intervallo di tempo `delta_t = 0.01 s` e, ad ogni passo, aggiornare la velocità, calcolando una variazione di $a\Delta t$ , e lo spostamento, calcolando una variazione di $v\Delta t$ . [P6.38]