Esercitazione di Laboratorio 09

Temi trattati

1. L'uso di costrutti condizionali per prendere decisioni (Ripasso Lab03)
2. Uso di cicli per l'esecuzione ripetuta di istruzioni (Ripasso Lab04)
3. Definizione ed elaborazione di liste e tabelle (Ripasso Lab08)

Discussione

A. Come si usano i costrutti condizionali per prendere decisioni complesse?
B. Una decisione complessa si può basare su condizioni generali e più specifiche. Come le deve gestire il costrutto condizionale?
C. Nella programmazione di cicli, cos'è un `off-by-one error`?
D. Quale è l'algoritmo di ricerca che serve per trovare un elemento in un insieme di elementi non ordinato?
E. Quale costrutto è il più adatto per scorrere i valori di una tabella con righe e colonne?

Esercizi

Parte 1 – Esercizi riepilogativi

***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.*

09.1.1 Lista di funzioni. Scrivere una funzione per ciascuna delle seguenti operazioni su liste di numeri interi, e fornire un programma per collaudare (*testing*) ciascuna funzione. Tutte le operazioni devono modificare la lista passata come parametro. [P6.4]

I. Scambiare tra loro il primo e l'ultimo elemento della lista.
II. Far slittare tutti gli elementi della lista di una posizione, spostando l'ultimo elemento nella prima posizione. Ad esempio, la lista `1 4 9 16 25` deve diventare `25 1 4 9 16`.
III. Sostituire con `0` tutti gli elementi di valore pari.
IV. Sostituire ciascun elemento, tranne il primo e l'ultimo, con il maggiore dei due elementi ad esso adiacenti.
V. Eliminare l'elemento centrale della lista se questa ha dimensione dispari, altrimenti eliminare i due elementi centrali.
VI. Spostare tutti gli elementi pari all'inizio della lista, lasciando quelli dispari in coda, e preservando l'ordinamento relativo tra gli elementi.
VII. Restituire il secondo valore maggiore della lista.
VIII. Restituire `True` se e solo se la lista è ordinata in senso crescente.
IX. Restituire `True` se e solo se la lista contiene due elementi adiacenti duplicati.
X. Restituire `True` se e solo se la lista contiene elementi duplicati (non necessariamente adiacenti).

09.1.2 Regole nascoste. Data una lista vuota `l = []`, scrivere un programma che la riempia con ciascuna delle sequenze di valori seguenti, dopo aver compreso la regola di generazione della sequenza (se esiste). [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 Diagramma a barre. Scrivere un programma che riceva in input una sequenza di valori reali e visualizzi un grafico a barre che li rappresenti, usando asterischi (`*`) per disegnare le barre. L'output deve seguire il seguente formato:

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

Assumere che tutti i valori nella sequenza di input siano positivi. Come primo passo, identificare il valore massimo. La barra che lo rappresenta deve essere composta di `40` asterischi. Le barre più corte devono usare un numero di asterischi proporzionale a questo per rappresentare i valori restanti. [P6.24]

09.1.4 E i numeri negativi? Estendere il programma dell'esercizio **09.1.3** affinché funzioni correttamente anche con sequenze che possono contenere numeri negativi. [P6.25]

09.1.5 Didascalie. Estendere il programma dell'esercizio **09.1.3** aggiungendo una didascalia a fianco di ciascuna barra. Il programma deve richiedere in input all'utente sia le didascalie che i valori della sequenza. L'output deve seguire il seguente formato:

```
  Egitto   **********************
 Francia   ****************************************
Giappone   ****************************
 Uruguay   *************************
Svizzera   **************
```

[P6.26]

09.1.6 Elenco di numeri interi. Scrivere un programma che acquisisca dall'utente un elenco di numeri interi positivi forniti su un'unica riga e separati dal carattere `':'`, ad esempio: `3:12:21:8:4:7`. Il programma deve stampare, mantenendo lo stesso formato:

I. i numeri inseriti, esclusi il minimo e il massimo (ad esempio, `12:8:4:7`);
II. tra i numeri inseriti, i soli numeri pari (ad esempio, `12:8:4`);
lll. tra i numeri inseriti, i soli numeri di 2 cifre (ad esempio, `12:21`).

Parte 2 – Applicazioni

***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.*

09.2.1 A teatro. La mappa dei posti a teatro è rappresentata da una tabella con i prezzi dei biglietti per ciascun posto, come questa.

```
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
```

Scrivere un programma che chieda all'utente di scegliere o un posto (fornendo riga e colonna), o un prezzo o l'uscita dal programma. Quando l'utente specifica un posto, accertarsi che sia libero e che le coordinate siano all'interno della tabella. Quando, invece, specifica un prezzo, assegnare un posto qualsiasi tra quelli disponibili a quel prezzo (se ve ne sono). Contrassegnare con un prezzo uguale a `0` i posti già venduti. [P6.27]

09.2.2 Parole concatenate. Nei lunghi viaggi in treno, per ingannare il tempo, si può fare il gioco delle "parole concatenate". Il primo giocatore dice una parola iniziale, poi a turno ciascun giocatore dovrà dire una parola non ancora detta durante la partita, la cui sillaba iniziale sia uguale alla sillaba finale della parola precedente. Un esempio di sequenza è: `'gatto', 'torino', 'notte', 'tela', 'lana'`. Per semplicità, ipotizziamo che tutte le sillabe siano lunghe esattamente `2` caratteri. Quindi, ad esempio, per `'figli'` la sillaba finale sarà `'li'` e non `'gli'`.
Scrivere un programma per permettere di gestire una o più partite del gioco. Ciascuna partita termina quando uno dei giocatori inserisce una parola già detta durante stessa partita, quando inserisce una parola concatenata in modo scorretto, oppure quando non riesce a inserire una nuova parola. Per abbandonare la partita, il giocatore di turno deve inserire un asterisco (`*`).

09.2.3 Il miglior cliente. Un supermercato vuole ricompensare il proprio miglior cliente del giorno, mostrandone il nome su uno schermo all'interno del negozio. A questo scopo, vengono memorizzati in una lista (`customers`) i nomi di tutti i clienti del giorno e, in un'altra lista (`sales`), l'importo della spesa effettuata da ciascuno di loro. Scrivere la funzione `name_of_best_customer(sales, customers)` che restituisca il **nome** del cliente che ha speso la cifra più alta. Poi, scrivere un programma che chieda al cassiere di digitare tutti gli importi spesi e i nomi dei relativi clienti, aggiungendoli, dopo ciascuna acquisizione, a due liste distinte, per poi invocare la funzione progettata e visualizzare il risultato. Usate il l'importo `0` come sentinella. [P6.33]

09.2.4 Spirale. Scrivere un programma che chieda all'utente di inserire un valore `N`, crei una matrice di `N x N` elementi, e vi inserisca una sequenza di numeri interi con valori compresi tra `1` e `N**2` visualizzando poi la tabella così riempita. Ad esempio, se `N = 4`, il programma dovrà stampare la tabella illustrata sulla sinistra, seguendo le direzioni illustrate nella tabella di destra (che non deve essere visualizzata dal programma).

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

⊙	→	→	↴
↱	→	↴	↓
↑	⊗	↲	↓
↑	←	←	↲

09.2.5 Negozio di animali. Un negozio di animali vuole fare uno sconto ai propri clienti se comprano uno o più animali ed almeno cinque altri prodotti. Lo sconto equivale al 20% del costo degli altri prodotti, esclusi gli animali. Scrivere una funzione `discount(prices, is_pet)` che riceva informazioni su una singola vendita e calcoli il valore dello sconto da applicare. Per il prodotto `i`-esimo, `prices[i]` è il prezzo prima di applicare sconti, e `is_pet[i]` è una variabile booleana che assume valore `True` se il prodotto `i`-esimo è un animale. Scrivere un programma che richiede a chi lavora in cassa di inserire in input, per ciascun prodotto venduto, il prezzo e poi, `'Y'` se il prodotto è un animale, e `'N'` se non lo è. Usare il prezzo `-1` come valore sentinella. Salvare gli input in una lista, chiamare la funzione implementata, e visualizzare lo sconto. [P6.32]

09.2.6 Alluvione. Si supponga di disporre di una tabella `heights` di $N \times N$ valori reali che riporta l'altitudine di un terreno nei diversi punti di un'area quadrata. Scrivere una funzione `flood_map(heights, water_level)` che, elaborando una tabella `heights` di $N \times N$ di numeri reali che rappresentano le altitudini, e dato il valore `water_level` del livello dell'acqua, costruisca e restituisca la mappa di una eventuale inondazione; tale mappa è rappresentata da una tabella $N \times N$ di stringhe di un carattere ciascuna (asterisco `'*'` per indicare che la casella è inondata, o spazio vuoto `' '` per indicare che la casella è all'asciutto).

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

Poi, scrivere un programma che costruisca una matrice `10x10` di altezze, leggendo `100` valori di altitudine del terreno, e visualizzi, richiamando la funzione precedente, come il terreno viene inondato quando il livello dell'acqua cresce. A tal fine, si divida l'intervallo di altezze (dalla minima alla massima) in `10` intervalli di uguale dimensione, e per ciascuno di questi 10 valori intermedi (`water_level`) si stampi la mappa della corrispondente inondazione. [P6.35]