Esercitazione di Laboratorio 04

Temi trattati

1. Cicli con gli enunciati `while` e `for`
2. Elaborazione di input complessi
3. Simulazioni

Discussione

A. Discutere somiglianze e le differenze tra i due gruppi di enunciati seguenti:
   a. `if`, `elif`, e `else`;
   b. `while` e `for`.
B. Cos'è un ciclo infinito?
C. Cos'è una simulazione?

Esercizi

Parte 1 – Cicli di base

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

04.1.1 Numeri interi. Scrivere un programma che legge una sequenza di numeri interi (la sequenza termina quando viene inserita una stringa vuota) e che, dopo ogni nuova acquisizione, calcoli e visualizzi:

I. le somme parziali di tutti i numeri acquisiti; Il programma deve visualizzare il risultato dei calcoli dopo ogni nuova acquisizione.
   Se, ad esempio, i valori in ingresso sono `1 7 2 9`, il programma visualizzerà la somma parziale dei numeri acquisiti dopo ogni acquisizione:
   a. alla prima acquisizione (`1`), il primo valore acquisito: `1`;
   b. alla seconda acquisizione (`7`), la somma tra la prima e la seconda acquisizione: `1 + 7 = 8`;
   c. alla terza acquisizione (`2`), la somma tra la prima, la seconda e la terza acquisizione: `1 + 7 + 2 = 10`;
   d. alla quarta acquisizione (`9`), la somma tra la prima, la seconda, la terza e la quarta acquisizione: `1 + 7 + 2 + 9 = 19`.
II. il valore minimo e il valore massimo tra quelli acquisiti;
III. il numero di valori pari e il numero di valori dispari tra quelli acquisiti. [P4.2]

04.1.2 Analisi di una stringa. Scrivere un programma che legga una riga di testo in ingresso sotto forma di stringa e visualizzi quanto segue:

I. le sole lettere maiuscole della stringa;
II. le lettere in posizioni pari nella stringa;
III. la stessa stringa in cui, al posto delle vocali (maiuscole o minuscole), vi sia un underscore (_);
IV. il numero di cifre numeriche presenti nella stringa;
V. le posizioni di tutte le vocali presenti nella stringa. [P4.3]

04.1.3 Lati. Scrivere un programma che legga un numero intero `n` e visualizzi, usando asterischi (`*`), un quadrato e un rombo pieni, il cui lato abbia lunghezza `n`. Se, ad esempio, l'utente fornisce il numero `4`, il programma deve visualizzare:

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


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

[P4.22]

04.1.4 Parole al contrario. Scrivere un programma che legga una parola e visualizzi:

I. la parola al contrario. Se, ad esempio, l'utente fornisce la stringa `'Ciao'`, il programma deve visualizzare `'oaiC'` [P4.9]
II. le lettere maiuscole partendo dal fondo. Se, ad esempio, l'utente fornisce la stringa `'CiAo'`, il programma deve visualizzare `'AC'`.

04.1.5 Numeri primi. Scrivere un programma che chieda all'utente un numero intero e, visualizzi in output un messaggio che descriva se il numero è primo oppure no.

04.1.6 Numeri primi. Scrivere un programma che chieda all'utente un numero intero e, poi, visualizzi tutti i numeri primi minori o uguali a tale numero. Se, ad esempio, l'utente fornisce il numero `20`, il programma deve visualizzare:

```
2
3
5
7
11
13
17
19
```

[P4.17]

04.1.7 Parole a pezzi. Scrivere un programma che legga una parola e visualizzi tutte le sue sottostringhe, ordinate per lunghezza crescente. Se, ad esempio, l'utente fornisce la stringa `'rum'`, il programma deve visualizzare:

```
r
u
m
ru
um
rum
```

[P4.12]

04.1.8 Numeri duplicati. Scrivere un programma che legge una sequenza di numeri interi (la sequenza termina quando viene inserita una stringa vuota) e che, dopo ogni nuova acquisizione, calcoli e visualizzi solamente i numeri adiacenti duplicati.

Se, ad esempio, i valori in ingresso al punto IV sono `1 3 3 4 5 5 6 6 6 3`, il programma visualizzerà i numeri adiacenti ripetuti almeno due volte, al termine della sequenza di numeri uguali, quindi al momento dell'acquisizione del primo numero diverso:

I. alla prima acquisizione (`1`), nulla, perché il valore considerato è solo uno;
II. alla seconda acquisizione (`3`), nulla, perché i due valori adiacenti considerati (`1` e `3`) non sono duplicati, cioè uguali tra loro;
III. alla terza acquisizione (`3`), ancora nulla, perché il valore acquisito è il duplicato del valore precedente, dunque la sequenza di numeri adiacenti duplicati è iniziata, ma non è detto che sia terminata;
IV. alla quarta acquisizione (`4`), il valore `3`, in quanto i due valori adiacenti considerati (`3` e `4`) non sono duplicati; pertanto, la sequenza di numeri adiacenti duplicati è terminata, e il programma deve visualizzare il numero duplicato.

Secondo questa logica, se i valori acquisiti sono `1 3 3 4 5 5 6 6 6 3`, il programma visualizzerà, al termine delle sequenze di acquisizioni che rilevano duplicati adiacenti, dunque alla prima acquisizione di un numero diverso: `3 5 6`. [P4.2]

Parte 2 – Applicazioni dei cicli

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

04.2.1 Il gioco di Nim. In questo gioco, due giocatori prelevano alternativamente biglie da un mucchietto. Ad ogni mossa, un giocatore sceglie quante biglie prendere: almeno una e al massimo metà delle biglie disponibili. Il giocatore che prende l'ultima biglia perde la partita.

Scrivere un programma che consenta all'utente di giocare contro il computer. Il programma deve:

I. generare un numero intero casuale compreso tra `10` e `100` da usare come dimensione iniziale del mucchietto di biglie;
II. generare un numero intero casuale, o `0` o `1`, da usare per decidere se sarà l'utente o il computer a giocare per primo;
III. generare un numero intero casuale, o `0` o `1`, da usare per decidere se il computer giocherà in modo intelligente o stupido:
    a. giocando in modo stupido, ad ogni sua mossa il computer semplicemente preleva dal mucchietto un numero di biglie casuale (ma valido, cioè compreso tra `1` e `n/2`, se nel mucchietto sono rimaste `n` biglie);
    b. giocando in modalità intelligente, invece, preleva un numero di biglie tale che il numero di quelle che rimangono nel mucchio sia una potenza di due diminuita di un'unità, cioè `3`, `7`, `15`, `31` o `63`. Quest'ultima è sempre una mossa valida, tranne quando la dimensione del mucchio è uguale a una potenza di due diminuita di un'unità. In tal caso, il computer fa una mossa scelta a caso (ovviamente tra quelle valide).

Come potrete verificare sperimentalmente, il computer non può essere battuto quando gioca in modalità intelligente e fa la prima mossa, a meno che la dimensione iniziale del mucchio non sia `15`, `31` o `63`. Analogamente, un giocatore umano che faccia la prima mossa e conosca la strategia qui descritta è in grado di battere il calcolatore. [P4.23]

04.2.2 Diagnostica per immagini. Il decadimento radioattivo dei materiali radioattivi può essere modellato dall'equazione $A = A_0 e^{-\lambda t}$, dove A è la quantità di materiale al tempo t, A_0 è la quantità al tempo 0 , e λ è il tasso di decadimento. In particolare $\lambda = \frac{\ln 2}{T_{1/2}}$, dove $T_{1/2}$ è l'emivita della sostanza (espressa nella stessa unità di misura di t).
Il Tecnezio-99 è un radioisotopo che viene usato per la diagnostica per immagini del cervello. Ha un'emivita di 6 ore. Scrivere un programma che visualizza la quantità relativa A/A_0 nel corpo di un paziente per ciascuna ora durante le 24 ore successive alla ricezione della dose. [P4.39]

04.2.3 Traiettorie. Supponiamo che una palla di cannone sia proiettata in aria con una velocità iniziale v_0. Qualunque libro di fisica affermerà che la posizione della palla dopo t secondi è $s(t) = -\frac{1}{2} g t^2 + v_0 t$, dove $g = 9.81\ m/s^2$ è l'accelerazione di gravità della Terra. Nessun libro di matematica menziona però perché qualcuno dovrebbe avere intenzione di procedere con questo esperimento chiaramente pericoloso; quindi, noi lo svolgeremo in totale sicurezza, con l'aiuto del computer. Infatti, confermeremo la teoria con l'aiuto di una simulazione. Nella nostra simulazione, considereremo come la palla si sposta in intervalli di tempo molto piccoli Δt . In un piccolo intervallo di tempo, la velocità v è praticamente costante, e possiamo calcolare la distanza percorsa dalla palla come $\Delta s = v \Delta t$. Scrivere un programma che imposti la costante `DELTA_T = 0.01` (secondi) e aggiorni la posizione della palla con:

```
s = s + v * DELTA_T.
```

La velocità cambia costantemente, in quanto viene ridotta dalla forza gravitazionale della Terra. In un piccolo intervallo di tempo, $\Delta v = -g \Delta t$, dunque, dobbiamo tenere la velocità aggiornata con:

```
v = v - g * DELTA_T.
```

In ciascuna iterazione successiva, il nuovo valore di velocità è utilizzato per aggiornare il calcolo della distanza percorsa. Il programma deve continuare l'esecuzione della simulazione fino a quando la palla di cannone cade a terra. Il programma prende in input la velocità iniziale (ad esempio, `100` metri al secondo). Aggiorna la posizione e la velocità `100` volte per ogni secondo simulato, ma visualizza in output la posizione calcolata solamente una volta per secondo simulato, insieme al valore calcolato dalla formula esatta $s(t) = -\frac{1}{2} g t^2 + v_0 t$, per confronto.

Nota: potrebbe sorgere un dubbio sul motivo per cui è vantaggioso simulare un fenomeno per cui esiste una formula esatta. Il motivo è che la formula non è veramente esatta. Ad esempio, l'accelerazione di gravità varia a seconda della distanza dalla Terra. Questo non è rappresentato dalla formula, mentre la nostra simulazione potrebbe essere estesa per includere questo aspetto. Per questo la simulazione si rende necessaria per le traiettorie di oggetti che volano più in alto, ad esempio i missili. [P4.37]