progetti_python

Nicolò Trebino

April 6, 2025

Contents

1 Calcolatrice Interattiva in Python

1.1 Obiettivo del progetto

Realizzare una **calcolatrice testuale interattiva** in Python che permetta all'utente di eseguire operazioni matematiche di base da terminale.

1.2 Requisiti minimi

Il programma deve:

1. Mostrare un menu interattivo con le seguenti opzioni:
   - Addizione
   - Sottrazione
   - Moltiplicazione
   - Divisione (gestire il caso di divisione per zero)
   - Potenza (esponenziale)
   - Uscita dal programma
2. Permettere all'utente di **selezionare un'operazione**, inserire i numeri richiesti e vedere il risultato a schermo.
3. Continuare a funzionare finché l'utente non sceglie di **uscire dal programma**.
4. Utilizzare **funzioni separate** per ogni operazione matematica.

1.3 Suggerimenti

- Usa `input()` per ricevere i dati dall'utente.
- Usa `float()` o `int()` per convertire i numeri.
- Puoi usare la libreria `math` per altre operazioni opzionali.

1.4 Estensioni opzionali (bonus)

Per chi vuole fare un passo in più:

1. Aggiungi altre operazioni come:
   - Radice quadrata (gestire il caso di numeri negativi)
   - Logaritmo
   - Fattoriale
   - Calcolo dell'M.C.D. o dell'm.c.m.

2 Gioco dell'Impiccato in Python

2.1 Obiettivo del progetto

Realizza in Python una versione testuale del classico gioco dell'**Impiccato**, dove l'utente deve indovinare una parola misteriosa inserendo una lettera alla volta.

2.2 Requisiti minimi

Il programma deve:

1. Scegliere casualmente una parola da indovinare da una lista predefinita (es. `["python", "programmazione", "informatica", "algoritmo", ...]`).
2. Visualizzare inizialmente la parola nascosta con degli **underscore** (_) al posto delle lettere.
3. Far inserire all'utente una lettera per volta, controllando che:
   - sia un carattere valido (una sola lettera)
   - non sia già stata inserita in precedenza
4. Verificare se la lettera è presente nella parola:
   - Se sì, aggiornare la parola mostrata all'utente
   - Se no, diminuire il numero di tentativi disponibili
5. Mostrare a ogni turno:
   - Le lettere indovinate finora
   - Il numero di tentativi rimanenti
6. Terminare il gioco quando:
   - L'utente ha indovinato tutta la parola (vittoria)
   - L'utente ha esaurito i tentativi (sconfitta)

2.3 Suggerimenti

- Usa `random.choice()` per selezionare una parola casuale dalla lista.
- Per gestire la parola nascosta, usa una lista di caratteri (`["_", "_", "_", ...]`) da aggiornare durante il gioco.
- Tieni traccia delle lettere già provate in una lista separata.
- Puoi usare `.lower()` per convertire le lettere in minuscolo e semplificare i controlli.

3 Lista della Spesa Interattiva in Python

3.1 Obiettivo del progetto

Creare un programma in Python che consenta all'utente di **gestire una lista della spesa** da terminale: visualizzare, aggiungere, rimuovere e ordinare elementi.

3.2 Requisiti minimi

Il programma deve:

1. Visualizzare un **menu testuale** con le seguenti opzioni:
   - Visualizza la lista
   - Aggiungi un elemento
   - Rimuovi un elemento
   - Ordina la lista
   - Esci
2. Permettere all'utente di **aggiungere elementi** alla lista, accettando solo parole alfanumeriche.
3. Permettere di **rimuovere elementi** specificando il nome esatto (verificando che siano presenti nella lista).
4. Mostrare la **lista attuale** ogni volta che l'utente lo richiede.
5. Ordinare la lista **in ordine alfabetico** su richiesta dell'utente.
6. Continuare a funzionare finché l'utente non seleziona l'opzione "Esci".
7. Utilizzare **funzioni separate** per ogni operazione (aggiunta, rimozione, visualizzazione, ordinamento).

3.3 Suggerimenti

- Usa una **lista vuota** per inizializzare la lista della spesa: `lista_spesa = []`.
- Puoi usare `.append()`, `.remove()` e `.sort()` per modificare la lista.
- Utilizza `input()` per ricevere i comandi dell'utente e `if/elif` per gestire il menu.
- Usa i **cicli** `while` per mantenere attivo il programma fino alla scelta dell'uscita.

3.4 Estensioni opzionali (bonus)

Per chi vuole aggiungere funzionalità extra:

1. Permettere all'utente di **salvare la lista in un file '.txt'**.

4 Sistema di Registrazione e Login in Python

4.1 Obiettivo del progetto

Realizzare un programma in Python che permetta all'utente di **registrarsi, accedere** con le proprie credenziali, **visualizzare e modificare** le proprie informazioni personali.

4.2 Requisiti minimi

Il programma deve:

1. Mostrare un **menu iniziale** con le seguenti opzioni:
   - Registrazione
   - Login
   - Uscita
2. Implementare una **funzione di registrazione** in cui l'utente inserisce:
   - uno username
   - una email
   - una password

   Le informazioni vengono salvate in una struttura dati.
3. Consentire il **login** solo se l'utente si è già registrato:
   - Controllare che email e password inseriti coincidano con quelli registrati
   - Se corretti, accedere a un secondo menu
   - Altrimenti mostrare un messaggio d'errore
4. Dopo il login, mostrare un **menu utente** con le seguenti opzioni:
   - Visualizza le informazioni personali (con password oscurata da asterischi `*`)
   - Cambia email
   - Cambia password
   - Logout
5. Il programma deve **continuare a funzionare** finché l'utente non seleziona "Esci".
6. Utilizzare **funzioni separate** per ogni azione

4.3 Suggerimenti

- Usa una **lista** o un dizionario per memorizzare le informazioni dell'utente (`[username, email, password]`).
- Puoi usare una variabile booleana per sapere se l'utente ha effettuato il login.
- Attenzione a controllare che i campi inseriti non siano vuoti e siano del tipo corretto.

4.4 Estensioni opzionali (bonus)

- Verifica di email nel formato corretto (es. con `@` e `.it` o `.com`).
- Permettere di **eliminare l'account**.
- Salvare e leggere i dati degli utenti da un **file di testo** per mantenere i dati tra un'esecuzione e l'altra.

5 Gestione della Rubrica Telefonica in Python

5.1 Obiettivo del progetto

Creare un programma in Python che consenta di gestire una **rubrica telefonica**. L'utente potrà aggiungere, cercare ed eliminare contatti, mantenendo i dati in memoria durante l'esecuzione del programma.

5.2 Requisiti minimi

Il programma deve:

1. Utilizzare un dizionario per **memorizzare i contatti**, dove la chiave è il nome del contatto e il valore è il numero di telefono.
2. Implementare un **menu interattivo** che consenta all'utente di scegliere tra le seguenti opzioni:
   - Aggiungere un contatto
   - Cercare un contatto
   - Eliminare un contatto
   - Uscire dal programma
3. **Funzione per aggiungere un contatto**:
   - Chiedere il nome e il numero di telefono
   - Aggiungere il contatto al dizionario (con nome come chiave e numero come valore)
4. **Funzione per cercare un contatto**:
   - Chiedere il nome da cercare
   - Visualizzare il numero associato al nome (se il contatto esiste nella rubrica)
5. **Funzione per eliminare un contatto**:
   - Chiedere il nome da eliminare
   - Rimuovere il contatto dalla rubrica se esiste
6. **Continuare a funzionare** finché l'utente non seleziona l'opzione "Exit".

5.3 Estensioni opzionali (bonus)

Per chi vuole estendere o migliorare il programma:

1. **Salvare la rubrica su file** (ad esempio un file '.txt' o '.json') per conservare i contatti tra le esecuzioni del programma.
2. **Visualizzare tutta la rubrica** (elenco completo di tutti i contatti) con un comando aggiuntivo.
3. Aggiungere un **controllo per i numeri di telefono** (ad esempio, assicurarsi che siano composti solo da numeri e abbiano una lunghezza valida).

6 Gestione delle Spese in Python

6.1 Obiettivo del progetto

Creare un programma in Python che consenta di registrare, visualizzare, filtrare e calcolare il totale delle spese, categorizzandole per tipo (es. cibo, trasporti, ecc.).

6.2 Requisiti minimi

Il programma deve:

1. Utilizzare una **lista di dizionari** per memorizzare le spese. Ogni spesa deve contenere le seguenti informazioni:
   - **Data** (formato 'GG/MM/AAAA')
   - **Importo** (numero decimale)
   - **Categoria** (ad esempio: cibo, trasporti, shopping, ecc.)
2. **Funzione per aggiungere una spesa**:
   - L'utente deve inserire la data, l'importo e la categoria della spesa.
   - La spesa deve essere salvata nel dizionario e aggiunta alla lista delle spese.
3. **Funzione per visualizzare tutte le spese**:
   - Mostrare tutte le spese registrate, con la data, l'importo e la categoria. Se non ci sono spese registrate, mostrare un messaggio adeguato.
4. **Funzione per calcolare il totale delle spese**:
   - Calcolare e stampare la somma totale di tutte le spese.
5. **Funzione per filtrare le spese per categoria**:
   - L'utente può cercare le spese appartenenti a una specifica categoria. Se non ci sono spese per quella categoria, il programma deve notificare l'utente.
6. Implementare un **menu interattivo** che permetta all'utente di:
   - Aggiungere una spesa
   - Visualizzare tutte le spese
   - Calcolare il totale delle spese
   - Filtrare le spese per categoria
   - Uscire dal programma
7. **Continuare a funzionare** finché l'utente non seleziona l'opzione "Esci".

6.3 Suggerimenti

- Usa una **lista di dizionari** per memorizzare le spese. Ogni dizionario rappresenterà una spesa e avrà le chiavi `"data"`, `"importo"` e `"categoria"`.
- Utilizza la **funzione 'sum()'** per calcolare il totale delle spese.
- Aggiungi un controllo per **validare l'importo** inserito, assicurandoti che sia un numero valido.
- Usa un ciclo `while` per mostrare il menu e raccogliere l'input dell'utente.

6.4 Estensioni opzionali (bonus)

Per chi vuole migliorare o estendere il progetto:

1. **Salvare le spese su un file** (ad esempio un file ‘.txt‘ o ‘.json‘) per mantenere i dati tra una sessione e l’altra.
2. **Visualizzare spese in ordine cronologico**, ordinando la lista delle spese per data.
3. Aggiungere un’opzione per **modificare una spesa** già inserita.
4. **Generare un report** che mostri il totale delle spese per ogni categoria.
5. **Creare un grafico** (utilizzando una libreria come ‘matplotlib‘) che mostri la distribuzione delle spese per categoria.

7 Consegna

Per chi vuole, è presente nella classe Moodle, Informatica Rapallo - TrebinoN, un form nel quale potete caricare il vostro codice.
È un’opportunità, soprattutto per chi volesse portare questo progetto come *capolavoro*; io guarderò il vostro codice e se dovessero esserci problemi ve li posso segnalare per una eventuale correzione.

7.1 Istruzioni per la consegna

1. Commentare il codice in modo appropriato.

   es.

   ```
   # Funzione che aggiunge alla lista "rubrica" un
      nuovo numero di telefono
   def aggiungi_telefono(numero):
       . . .
   ```

2. Utilizzate il `markdown` nel notebook per indicare quale progetto avete scelto e per descrivere eventuali estensioni opzionali o idee personali che avete deciso di implementare.
3. Caricare nel form della classe Moodle solamente il python notebook: il file `.ipynb`.
4. Sarebbe gradito se mi scriveste una mail così da avvisarmi che avete caricato il file, altrimenti potrei ”pescarmelo”.

   Clicca qui per inviare la mail