Lezione S2/L5

Linguaggio python(3)

La Lezione di oggi ha richiesto di analizzare un programma scritto in linguaggio python. in questo esercizio dovevamo:

- Capire cosa fa il programma senza eseguirlo.
- Individuare nel codice sorgente le casistiche non standard che il programma non gestisce (esempio, comportamenti potenziali che non sono stati contemplati).
- individuare eventuali errori di sintassi / logici.
- Proporre una soluzione per ognuno di essi.

```
import datetime

def assistente_virtuale(comando):

        if comando =="Qual è la data di oggi?":

                oggi = datetime.datetoday()

                risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")

        elif comando =="Che ore sono?":

                ora_attuale = datetime.datetime.now().time()

                risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")

        elif comando =="Come ti chiami?":

                risposta="Mi chiamo Assistente Virtuale"

        else:

                risposta = "Non ho capito la tua domanda."

        return risposta


while True
        comando_utente=input("Cosa vuoi sapere? ")

        if comando_utente.lower()=="esci":

                print("Arrivederci")
                break

        else:

                print(assistente_virtuale(comando_utente))
```

figura 1: Il codice

Analizziamo ora le varie parti del codice e capiamo come funziona. Il programma esegue le seguenti operazioni:

1. **Importazione della Libreria `datetime`**
   La libreria `datetime` viene importata per consentire la gestione e la formattazione di date e orari.

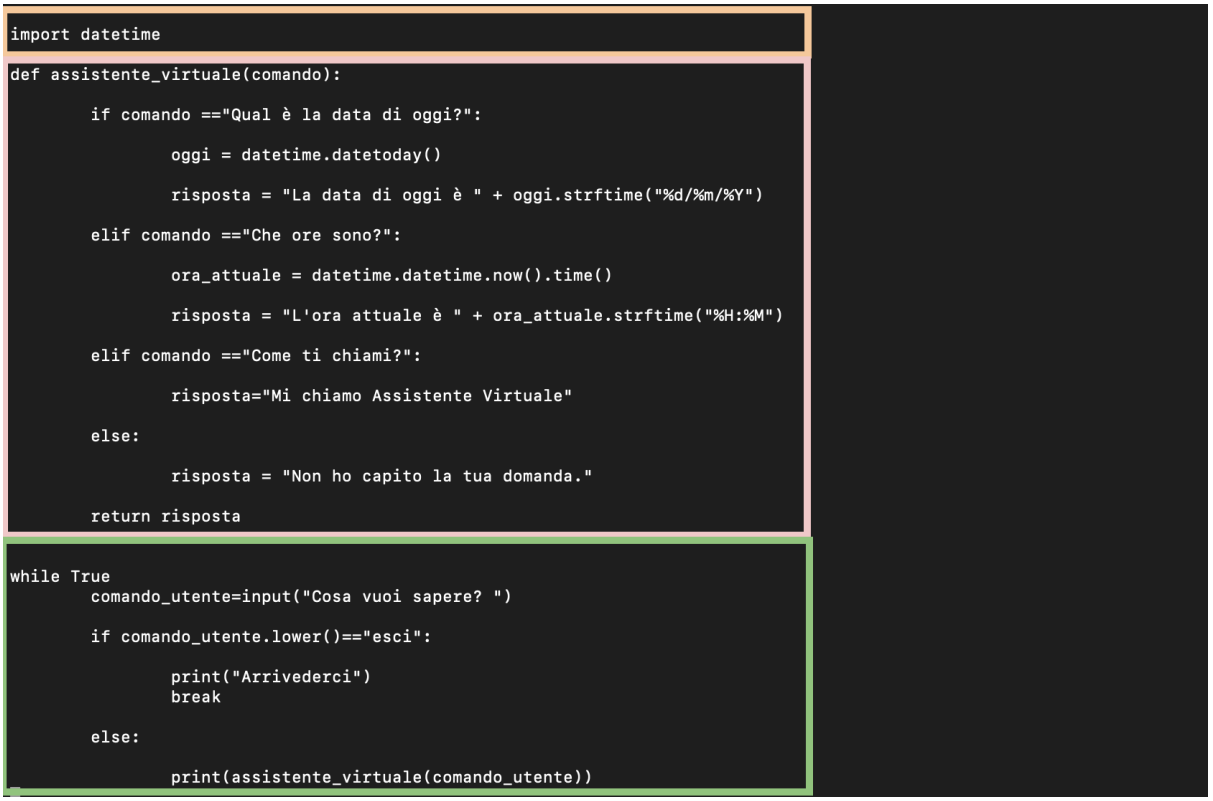
2. **Definizione della Funzione `assistente_virtuale`**
   La funzione `assistente_virtuale` prende in ingresso una variabile denominata

`comando`. In base al valore del comando, attraverso un'istruzione condizionale `if` con più casi (`elif`), restituisce risposte diverse:

- **Data odierna**: Chiedendo "*Qual è la data di oggi?*", si utilizza `datetime.datetoday()` per ottenere la data attuale e il metodo `.strftime()` per formattarla come stringa in un formato leggibile (ad esempio, `giorno/mese/anno`).
- **Ora attuale**: Chiedendo "*Che ore sono?*", si utilizza `datetime.datetime.now().time()` per ottenere l'ora corrente. Viene utilizzato `.strftime()` per formattare l'orario in un formato leggibile (`H:M`). È presente un'osservazione sul fatto che `datetime` venga ripetuto due volte (`datetime.datetime`), ma non si tratta di un errore bensì del modo in cui la libreria `datetime` è strutturata, infatti il primo datetime è per chiamare il modulo il secondo invece è per la classe.
- **Nome dell'assistente virtuale**: Chiedendo "*Come ti chiami?*", restituisce una risposta predefinita indicando il nome dell'assistente virtuale.
- **Comando non riconosciuto**: Se il comando fornito non corrisponde a nessuna delle opzioni precedenti, viene restituito un messaggio che comunica che la domanda non è stata compresa.

3. **Ciclo Principale del Programma**
   Il programma è racchiuso in un ciclo `while` con condizione `True`, che lo rende un ciclo infinito finché non viene espressamente interrotto. Ad ogni iterazione:
   - L'utente inserisce una domanda tramite `input`.
   - Se il comando inserito è "`esci`" (ignorando le maiuscole e minuscole grazie al metodo `.lower()`), il programma stampa un messaggio di arrivederci e termina l'esecuzione.
   - Per qualsiasi altro comando, la funzione `assistente_virtuale` viene chiamata, e la risposta generata viene mostrata all'utente.

Questo schema rende il programma interattivo e persistente, finché non viene esplicitamente chiuso (figura 2).



```
import datetime

def assistente_virtuale(comando):

        if comando =="Qual è la data di oggi?":

                oggi = datetime.datetoday()

                risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")

        elif comando =="Che ore sono?":

                ora_attuale = datetime.datetime.now().time()

                risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")

        elif comando =="Come ti chiami?":

                risposta="Mi chiamo Assistente Virtuale"

        else:

                risposta = "Non ho capito la tua domanda."

        return risposta

while True
        comando_utente=input("Cosa vuoi sapere? ")

        if comando_utente.lower()=="esci":

                print("Arrivederci")
                break

        else:

                print(assistente_virtuale(comando_utente))
```

figura 2: breakdown del codice

POSSIBILI PROBLEMI LOGICI

- Osservando il Programma balza subito all'occhio un problema condiviso da quasi tutte le richieste di input, infatti queste sono **case sensitive**, ovvero l'aggiunta di una lettera maiuscola non fanno riconoscere al programma l'effettiva richiesta (scrivere "qual è la data oggi?" da errore a causa della lettera maiuscola mancata Q). Oltretutto viene richiesta in input una stringa lunga, che aumenta la possibilità di errori di scrittura.
- L'utilizzo della funzione **datetime.datetime.now().time()** richiede di un'osservazione riguardo a cosa questa funzione fa specificatamente, strana la chiamata di now() seguita da time().
- **Mancanza di un legenda** di tutti i possibili comandi, un utente inesperto potrebbe pensare che il programma fornisca soltanto come risposta "Non ho capito la tua domanda".
- Sebbene il comando "esci" inserito dall'utente sia case insensitive rimane comunque suscettibile ad eventuali i**nserimenti di spazi alla fine** o all'aggiunta di punteggiatura come virgola e punti, questo punto **vale anche per tutti gli altri input**.

Dopo aver salvato il programma con "**ctrl +O**" possiamo uscire. Per poter svolgere il codice dobbiamo ora prima cambiare i permessi tramite **chmod** per renderlo eseguibile e poi senza necessità di creare un eseguibile digitiamo, **python3 ./VirtualAssistant.py**.

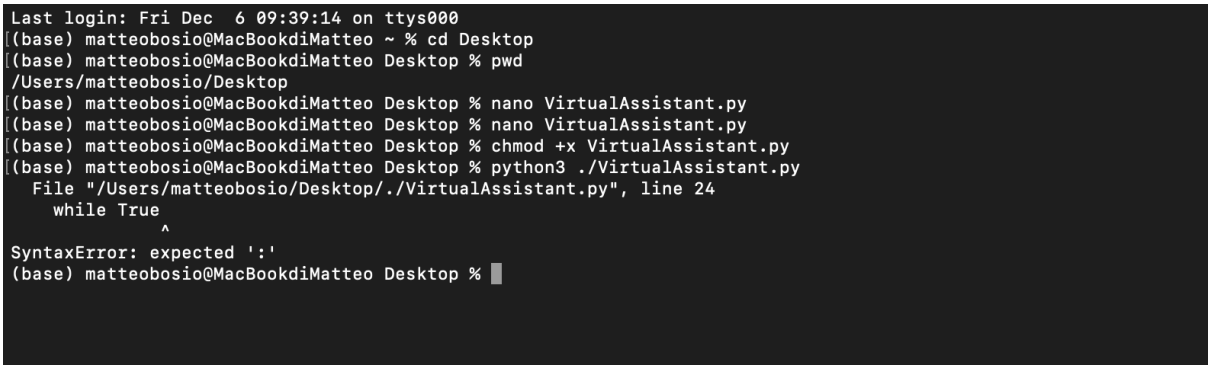


figura 3: Prima esecuzione del programma

come vediamo il programma contiene un errore sintattico dovuto alla mancanza dei due punti, entriamo nel codice, li aggiungiamo e eseguiamo ancora il programma.

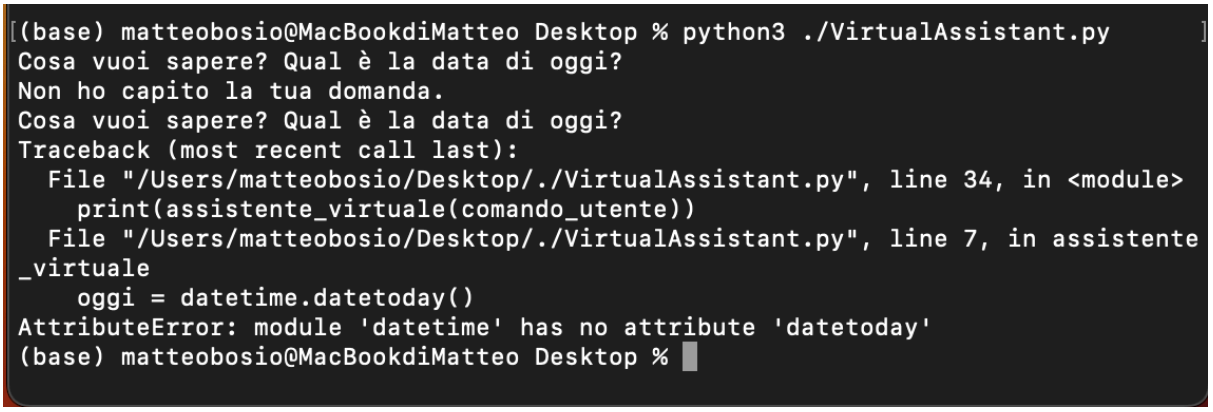


figura 4: Seconda esecuzione

Il programma ora viene eseguito ma appena proviamo a chiedere la data ci esce questo errore (figura 4), il testo ci dice che l'attributo .datetoday() non esiste, cercando nelle risorse online ho trovato che in realtà il comando corretto pone un punto tra date e today trasformandolo quindi in .date.today().
modifichiamo quindi il programma e lo eseguiamo una terza volta.

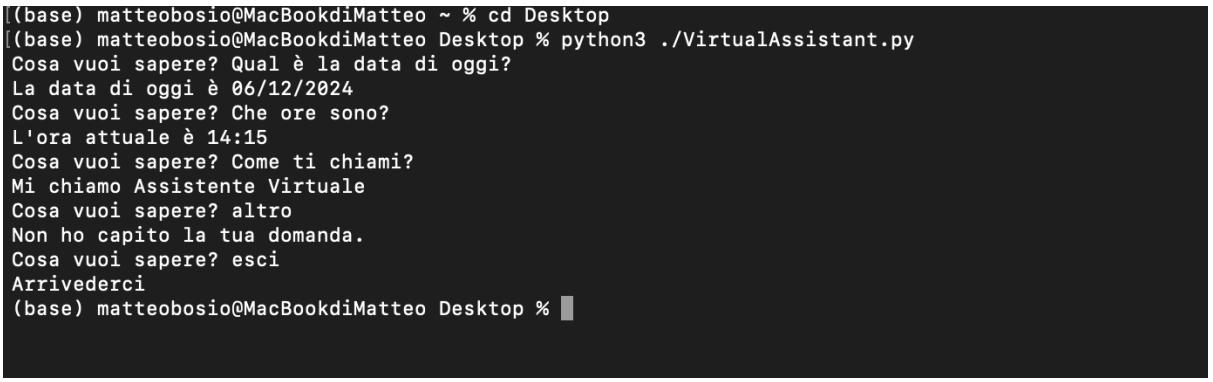


figura 5: Terza esecuzione

Come osserviamo ora il programma funziona correttamente ed è implementato in maniera corretta.

Per **migliorare** ancora di più il codice possiamo :

1. Usare .lower() o .strip() in tutti gli input per uniformare l'input prima di confrontarlo, ad esempio (if comando.lower().strip() == "qual è la data di oggi?":)
2. Fornire un elenco di comandi accettati, ad esempio all'inizio del programma:
   ```
   print("Benvenuto! Puoi chiedermi:")
   print("- Qual è la data di oggi?")
   print("- Che ore sono?")
   print("- Come ti chiami?")
   print("Scrivi 'esci' per terminare il programma.")
   ```
   oppure associare a questi comandi un numero così da ridurre al minimo i possibili errori di battitura.
3. Implementare un output più informativo in caso di errore, dire semplicemente "Non ho capito" non aiuta l'utente a migliorare il proprio input.
   ```
   print("Prova a vedere la lista di comandi possibili")
   ```
4. Cercando online ho osservato anche il fatto che la riga datetime.datetime.now().time() può essere scritta anche senza il .time() e il programma funziona correttamente infatti:

Aspect	`datetime.now()`	`datetime.now().time()`
Object Type	`datetime.datetime`	`datetime.time`
Contains	Date + Time (e.g., `2024-12-06 14:30:45`)	Only Time (e.g., `14:30:45`)
Usage	Good for working with both date and time	Good for working with only the time
Formatting	`strftime` works on the full object	`strftime` works on the time component

figura 6 : Diversi utilizzi delle funzioni

BONUS:

In questo esercizio bonus ci viene richiesto scrivere un programma Python per gestire una lista della spesa. Il programma deve permettere all'utente di:

1) Aggiungere un elemento alla lista.
2) Rimuovere un elemento dalla lista (se presente).
3) Visualizzare tutti gli elementi della lista ordinati in ordine alfabetico.
4) Salvare la lista su file.
5) Caricare una lista da file.

Il programma deve avere un menu che consente all'utente di scegliere le varie operazioni e deve terminare solo quando l'utente lo richiede.

```python
import os

def il_menu():
    print("\nMenu della lista della spesa")
    print("1 - Aggiungere un elemento alla lista")
    print("2 - Rimuovere un elemento dalla lista")
    print("3 - Visualizzare tutti gli elementi ordinati alfabeticamente")
    print("4 - Salvare la lista su file")
    print("5 - Caricare una lista da file")
    print("6 - Uscire dal programma")

def aggiungi_elemento(lista):
    elemento = input("Inserisci l'elemento da aggiungere: ").strip()
    if elemento:
        lista.append(elemento)
        print(f"'{elemento}' è stato aggiunto alla lista.")
    else:
        print("Elemento non valido.")

def rimuovi_elemento(lista):
    elemento = input("Inserisci l'elemento da rimuovere: ").strip()
    if elemento in lista:
        lista.remove(elemento)
        print(f"'{elemento}' è stato rimosso dalla lista.")
    else:
        print(f"'{elemento}' non è presente nella lista.")

def visualizza_lista(lista):
    if lista:
        print("\nLista della Spesa (in ordine alfabetico):")
        for elemento in sorted(lista):
            print(f"- {elemento}")
    else:
        print("La lista è vuota.")

def salva_su_file(lista, nome_file="lista_spesa.txt"):
    try:
        with open(nome_file, "w") as file:
            for elemento in lista:
                file.write(elemento + "\n")
        print(f"Lista salvata su '{nome_file}'.")
    except Exception as e:
        print(f"Errore durante il salvataggio: {e}")

def carica_da_file(nome_file="lista_spesa.txt"):
    try:
        if not os.path.exists(nome_file):
            print(f"Il file '{nome_file}' non esiste.")
            return []
        with open(nome_file, "r") as file:
            lista = [linea.strip() for linea in file]
        print(f"Lista caricata da '{nome_file}'.")
        return lista
    except Exception as e:
        print(f"Errore durante il caricamento: {e}")
        return []

def main():
    lista_spesa = []
    while True:
        il_menu()
        scelta = input("\nScegli un'opzione (1-6): ").strip()
        if scelta == "1":
            aggiungi_elemento(lista_spesa)
        elif scelta == "2":
            rimuovi_elemento(lista_spesa)
        elif scelta == "3":
            visualizza_lista(lista_spesa)
        elif scelta == "4":
            salva_su_file(lista_spesa)
        elif scelta == "5":
            lista_spesa = carica_da_file()
        elif scelta == "6":
            print("Uscita dal programma. Arrivederci!")
            break
        else:
            print("Scelta non valida. Riprova.")
```

figura 7: codice bonus

il codice presenta 6 diverse funzioni più una funzione main() non necessaria ma utile per tenere il codice più pulito.

- **LIBRERIA OS:** La libreria os viene utilizzata per operazioni legate al file system, come controllare l'esistenza di un file (os.path.exists). Necessaria nella funzione carica_da_file, dove si verifica se il file esiste prima di tentare di leggerlo.

- **MOSTRA_MENU:** Mostra un menu per guidare l'utente nella scelta delle operazioni disponibili. La struttura del menu è chiara e leggibile. Viene stampato ogni volta che il ciclo principale richiede un input, migliorando l'usabilità.

- **AGGIUNGI_ELEMENTO:** Permette di aggiungere un elemento alla lista della spesa. L'uso di .strip() rimuove eventuali spazi superflui, evitando che input come " pane " vengano salvati con spazi indesiderati. Verifica inoltre che l'input non sia vuoto.

- **RIMUOVI_ELEMENTO:** Rimuove un elemento dalla lista se presente. Controlla se l'elemento esiste nella lista prima di tentare di rimuoverlo, evitando errori. Comunica chiaramente all'utente se l'elemento non è trovato, migliorando l'esperienza d'uso.

- **VISUALIZZA_LISTA:** Mostra tutti gli elementi della lista ordinati alfabeticamente. Usa sorted() per presentare la lista in ordine alfabetico senza modificare l'ordine originale. Mostra un messaggio chiaro se la lista è vuota, evitando un'uscita vuota o confusa.

- **SALVA_SU_FILE:** Salva la lista corrente su un file di testo. Usa un blocco try per gestire errori, come problemi di permessi o spazio insufficiente. Salva ogni elemento della lista su una nuova riga, creando un file leggibile. Il file predefinito è lista_spesa.txt, ma può essere specificato un altro nome se necessario.

- **CARICA DA FILE:** Carica una lista da un file di testo. Controlla se il file esiste prima di tentare di aprirlo, prevenendo errori. legge e pulisce ogni riga del file restituendo una lista vuota in caso di errore o se il file non esiste.

- **MAIN:** Contiene il ciclo principale del programma e gestisce le interazioni con l'utente. Inizializza la lista e usa un ciclo while infinito per mantenere il programma attivo fino alla scelta di uscire. Mostra anche il menu e richiede una scelta all'utente. Gestisce infine input non validi con un messaggio di errore.

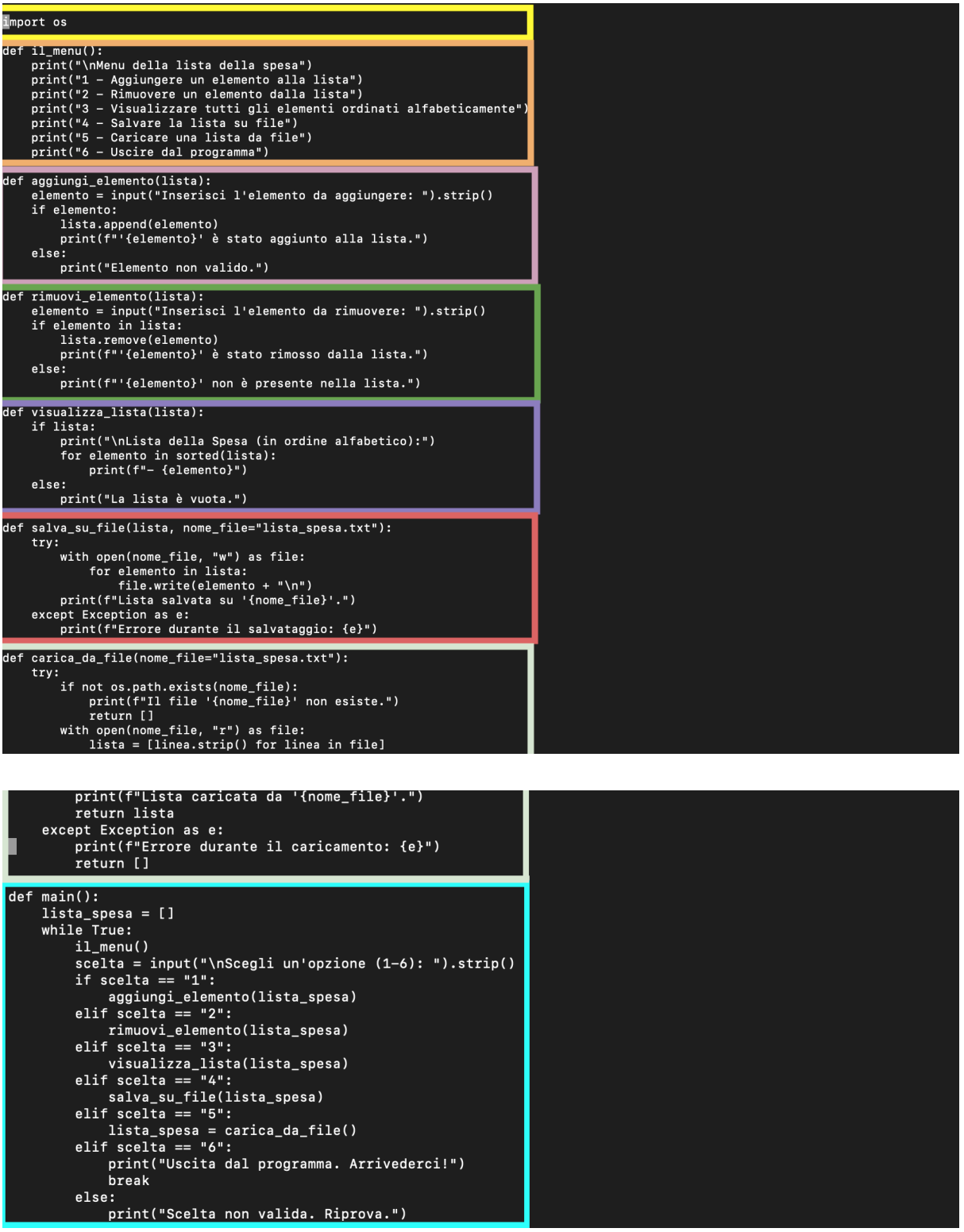


figura 8: breakdown del codice

La maggior parte del codice è complessivamente semplice eccetto le funzioni di salvataggio e caricamento del file, è proprio qua che subentra la libreria os. La maggior parte delle informazioni relative a questa libreria le ho ottenute online, in particolare sono interessanti le righe con **try e except.** Che Gestiscono potenziali errori durante l'operazione di scrittura come ad esempio:

- Permessi insufficienti per scrivere sul file.
- Spazio su disco insufficiente.
- File non accessibile.
- Formato del file incompatibile.

anche **file.write(elemento + "\n")** e **[linea.strip() for linea in file]** hanno rispettivamente il compito di:

- Scrive il contenuto della variabile elemento nel file, aggiungendo un ritorno a capo (\n) alla fine di ogni elemento.
- Legge ogni riga del file, rimuovendo eventuali spazi bianchi o caratteri di fine linea (\n) con .strip().

Interessante è anche os.path.exists(nome_file) che verifica se il file specificato esiste per prevenire errori sollevati dall'apertura di un file inesistente, mostrando un messaggio chiaro all'utente.

Complessivamente l'output del sistema è mostrato nelle seguenti immagini: (figura 9).
Lascio le immagini a dimensione piena per poterle osservare correttamente.

```
(base) matteobosio@MacBookdiMatteo Desktop % python3 ./ListaSpesa.py

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 1
Inserisci l'elemento da aggiungere: cereali
'cereali' è stato aggiunto alla lista.

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 1
Inserisci l'elemento da aggiungere: latte
'latte' è stato aggiunto alla lista.

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 2
Inserisci l'elemento da rimuovere: latte
'latte' è stato rimosso dalla lista.

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 3

Lista della Spesa (in ordine alfabetico):
- cereali
```

```
Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 4
Lista salvata su 'lista_spesa.txt'.

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 5
Lista caricata da 'lista_spesa.txt'.

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma

Scegli un'opzione (1-6): 3

Lista della Spesa (in ordine alfabetico):
- Acqua
- Cereali
- Latte
- Miele
- Pasta

Menu della lista della spesa
1 - Aggiungere un elemento alla lista
2 - Rimuovere un elemento dalla lista
3 - Visualizzare tutti gli elementi ordinati alfabeticamente
4 - Salvare la lista su file
5 - Caricare una lista da file
6 - Uscire dal programma
```

figura 9: output del programma