Report del progetto U1S2L5

Informazioni Generali

L’esercizio consiste nell’analisi di un programma in python.
Questo report sarà diviso in 3 parti, le prime due seguiranno i primi due quesiti dell’esercizio, ovvero una parte dove verrà spiegato cosa fa il programma ed un elenco delle casistiche non standard che il programma farebbe fatica a gestire. Nell’ultima parte invece verranno identificati gli errori e verrà proposta per essi una soluzione.

Cosa fa il programma?

Il programma in questione funziona come un assistente virtuale che può rispondere a domande definite in un ciclo interattivo:
Domande:

1. Qual è la data di oggi?
2. Che ore sono?
3. Come ti chiami?

Il programma è composto da:

- **Libreria (***datetime*): Il programma sfrutta la libreria standard Python datetime per recuperare informazioni aggiornate in tempo reale, in particolare l'ora e la data correnti del sistema.
- **Ciclo Interattivo** (*while True*): L'interazione con l'utente è gestita tramite un ciclo infinito (*while True*). Questo assicura che il programma rimanga attivo, accettando continuamente l'input dell'utente, finché non viene invocata la condizione di uscita.
- **Logica Condizionale** (*if...elif...else*): La gestione delle risposte è demandata a una struttura condizionale (*if...elif...else*). Questa blocca confronta l'input dell'utente con le domande prestabilite e **restituisce la risposta corrispondente** (sia essa una stringa fissa, un dato dinamico fornito da datetime o una combinazione delle due).

Condizione di terminazione:
L'esecuzione del programma termina esplicitamente solo quando l'utente immette un comando specifico (per questo programma 'esci'), che attiva l'istruzione break per uscire dal ciclo *while*.

Casistiche non standard

Nel codice sorgente sono presenti alcune casistiche non standard che il programma non riesce a gestire. Il programma presenta numerose carenze

nella User Experience. È fondamentale implementare misure che guidino l'utente e gestiscano le interazioni non previste.

1. Assenza di Istruzioni Iniziali e Contesto

Problema: L'utente non sa cosa aspettarsi o quali sono i limiti del programma.

Necessità di Chiarezza: È cruciale informare l'utente sulla **funzionalità esatta** del programma all'avvio. La **buona pratica** suggerisce di non dare per scontata la conoscenza dell'utente.

Inserire un semplice messaggio di benvenuto o un'istruzione print all'inizio del codice lo renderebbe più intuitivo.

2. Mancanza di Orientamento e Guida all'Input

L'utente non è guidato sui comandi validi, portandolo a inserire input che il programma non può gestire. Gli utenti tendono a "testare i limiti" di un programma. Senza indicazioni chiare (come un **menu** o un **elenco di comandi**) per l'utente sarà molto difficile, se non impossibile, indovinare le query esatte del programma. Inoltre, il programma non include nessuna logica per renderlo meno sensibile alle **maiuscole/minuscole**. Accettando le stringhe solo ed esattamente come vengono digitate, si potrebbe utilizzare almeno il metodo *.lower()* sull'input dell'utente.

Tuttavia, l'uso di *.lower()*, sebbene necessario, **non risolve** il problema degli **errori di battitura o spaziatura**, che manderebbero comunque l'input nel blocco else della funzione. Per eliminare inoltre i problemi di spaziatura, potremmo usare .split() per eliminare dall'input ogni carattere di spazio.

Ad esempio, potremmo utilizzare una funzione come nello screenshot per convertire l'input dell'utente.

```
# Rimuovere spazi bianchi e convertire in minuscolo
comando = comando_utente.strip().lower()
```

Tuttavia, per non complicare troppo il programma (e anche perché lo reputo più semplice), vengono proposte due soluzioni per guidare l'utente:

Menù di Scelta: Si definisce una variabile stampa_menu per incapsulare il codice originale, aggiungendo innanzitutto la possibilità per l'utente di comprendere lo scopo del programma. Verranno visualizzate a schermo tre opzioni che fungono da istruzioni per l'utente. A ogni opzione verrà abbinato un numero: in questo modo, l'utente potrà digitare sia il testo dell'opzione sia il numero corrispondente per visualizzare la query appropriata.

```python
def stampa_menu():
    print("\nCiao! Sono l'Assistente Virtuale. Scegli una domanda:")
    print("1. Qual è la data di oggi?")
    print("2. Che ore sono?")
    print("3. Come ti chiami?")
    print("Oppure digita 'esci' per uscire.")

def assistente_virtuale(comando_utente):


    if comando_utente == "Qual è la data di oggi?" or comando_utente == "1":
        oggi = datetime.datetime.today()
        risposta = "La data di oggi è " + oggi.strftime("%d/%m/%y")

    elif comando_utente == "Che ore sono?" or comando_utente == "2":
        ora_attuale = datetime.datetime.now().strftime("%H:%M")
        risposta = ("L'ora attuale è " + ora_attuale)

    elif comando_utente == "Come ti chiami?" or comando_utente == "3":
        risposta = "Mi chiamo Assistente Virtuale"

    else:
        risposta = "Non ho capito la tua domanda"

    return risposta
```

Fig. 1: *Esempio di menù di scelta*

Feedback nell'Output "*else*": Per un assistente virtuale basato su domande/risposte come questo, una soluzione molto efficiente è sfruttare il blocco "*else*" della funzione di risposta. Quando il comando non è riconosciuto, l'Assistente dovrebbe elencare immediatamente i **comandi validi** che può gestire.

```
else:

        # Suggerimento per migliorare l'usabilità

        risposta = "Non ho capito la tua domanda. I comandi
validi sono: 'data', 'ora', 'come ti chiami?' o 'esci'."
```

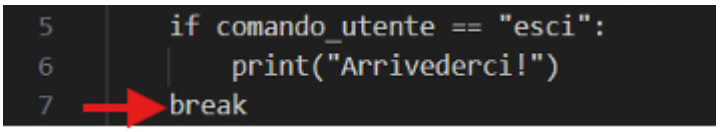
Fig. 2: *Esempio di feedback nell'output*

Identificazione e risoluzione degli errori

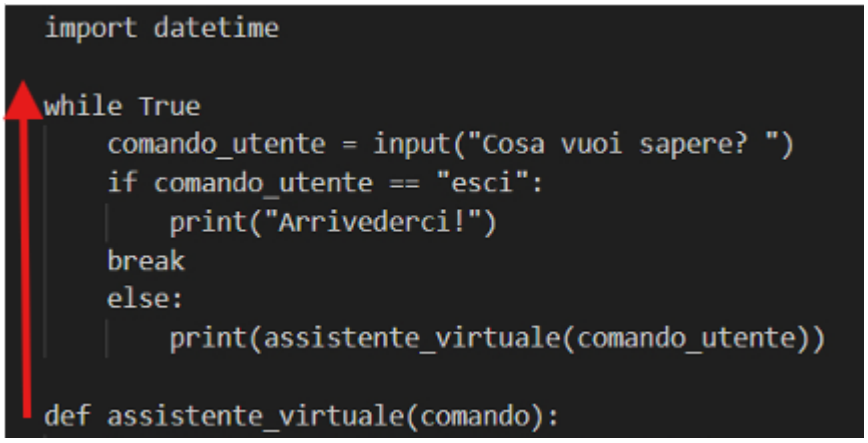
Nella riga 3, manca il carattere due punti (:) alla fine dell'istruzione while True. Questo è un errore di sintassi che impedisce l'avvio del programma. Aggiungere i due punti: per risolvere il problema.

```
3   while True
```

Nella riga 7, l'istruzione "break" è indentata allo stesso livello del blocco if (riga 5), il che la rende un'istruzione separata nel ciclo while. L'obiettivo di questo "break" è uscire solo quando si digita il comando "esci", in questo caso, va indentato correttamente sotto il blocco "*if*".



Nella riga 9 viene chiamata una funzione senza averla prima definita (nell'esercizio, viene definita a **linea 11**), essendo Python un linguaggio di programmazione che legge dall'alto verso il basso, non vedendo la funzione definita, non farà funzionare il programma. La soluzione logica sarebbe di spostare l'intera definizione della funzione all'inizio del file, subito dopo le importazioni della libreria, prima che inizi il ciclo "while".



Alla riga 13 viene inserito il comando datetime.datetoday(). Tuttavia, secondo la documentazione di Python, all'interno del modulo datetime, non esiste un metodo chiamato datetoday(). Il metodo corretto per ottene la data corrente è .today().
Il comando corretto sarà quindi datetime.datetime.today()

```
13          oggi = datetime.datetoday()
```

Alle righe 12, 15 e 18, viene utilizzata la variabile comando. Sebbene questo non sia un errore, io avrei preferito utilizzare “comando_utente” (**riga 11**) per avere un codice più omogeneo (entrambi definiti nei limiti della funzione, un’alternativa sarebbe definirli all’esterno della funzione).
Per garantire la correttezza del codice, basta infatti assicurarsi che la variabile utilizzata all’interno della funzione corrisponda al nome del parametro.

Giuseppe Monaco