Consegna:

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

codice sorgente (screen da discord):

```
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))
```

Funzione del codice

All'esecuzione del codice viene chiesto all'utente cosa vuole sapere e dopo aver ottenuto l'input da tastiera passa la domanda alla funzione "assistente_virtuale()" che genera una risposta stampata a video con la funzione print().

Errori sintattici e logici rilevati

1. Mancano i 2 punti dopo il while true.
2. La comparazione tra stringhe in python e' case-sensitive, trasformarle con la funzione lower() le renderà tutte diverse dalle possibili opzioni previste facendo eseguire solo il codice dell'else nella funzione assistente_virtuale().
   Una possibile soluzione e' trasformare subito la stringa usando lower() e togliendo le maiuscole dai controlli.
   es: `comando_utente = input("...").lower()`
3. La parola chiave "datetime" non esiste. per ottenere lo stesso risultato, va chiamata dal modulo datetime la classe date e poi il metodo today().
   `datetime.date.today()`
4. datetime.datetime.now().time() non presenta errori ma rappresenta un tempo senza fuso orario. questo puo' creare problemi se l'applicazione viene usata a livello internazionale. e' consigliato usare `datetime.now(timezone.utc)` per ridurre le incompatibilita'.

Potenziali comportamenti pericolosi dell'utente

1. In "comando_utente" potrebbe essere inserito un numero o un altro tipo di dato e non convertito a dovere da python, scatenando possibili bug e interruzioni durante la comparazione delle stringhe.
   Si consiglia in questo caso, di fare un casting esplicito:
   `comando_utente = str(input("..."))`
   Oppure usare try-except per avvisare l'utente di errori di input e evitare bug o interruzioni.
   la prima soluzione puo' comunque portare l'esecuzione del codice sempre sull'else della funzione assistente_virtuale()
   Una soluzione ottimale prevede entrambe le modifiche.

Codice sorgente corretto

```
import datetime
def assistente_virtuale(comando):
        if comando == "qual e' la data di oggi?":
                oggi = datetime.date.today()
                risposta = "la data di oggi e' " + oggi.strftime("%d/%m/%Y")
        elif comando == "che ore sono?":
                ora_attuale = datetime.datetime.now(datetime.timezone.utc).time()
                risposta = "l'ora attuale e' " + 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:
        try:
                comando_utente = str(input("cosa vuoi sapere?")).lower()
        except Exception as e:
                print("si e' verificato un errore durante l'inserimento:", e)
                continue

        if comando_utente == "esci":
                print("arrivederci!")
                break
        else:
                print(assistente_virtuale(comando_utente))
```

Ulteriori considerazioni / miglioramenti finali

1. L'utente potrebbe non conoscere i comandi disponibili. Si può aggiungere un comando "help" o "aiuto" che stampa i comandi disponibili (l'uso del comando dovrebbe essere consigliato all'inizio dell'esecuzione del codice).

2. Numerare i comandi e accettare in input solo il numero del comando potrebbe facilitare l’esperienza dell’utente e quella del programmatore, permettendogli di gestire i comandi come dizionario numero_comando - stringa_comando