ANALISI CODICE in PYTHON

"ASSISTENTE VIRTUALE"

INDICE

1) INTRODUZIONE

Il codice in analisi è il seguente:

```
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))
```

Possiamo notare, senza un'analisi approfondita, che è stata impostata una DEF (una parola chiave che definisce una funzione) "Assistente virtuale"; si tratta perciò di un programma in grado di chiedere all'utente cosa vuole sapere e, in base alla domanda posta dall'utente, fornire una risposta ben precisa.

E' stato inserito il ciclo WHILE, il che significa che l'azione verrà ripetuta ciclicamente fin quando l'utente da un comando specifico per interrompere la comunicazione.

Inoltre, come primo dato disponibile, abbiamo IMPORT DATETIME che va a definire quale libreria stiamo utilizzando per scrivere il codice; in questo caso si tratta di una libreria che consente di gestire operazioni per ottenere data e ora.

Di conseguenza **il codice dovrà generare un Assistente Virtuale in grado di rispondere a domande relative a data ed ora attuali.**

2) ANALISI DEL FUNZIONAMENTO

Per verificare la correttezza del codice è necessario eseguirlo, in modo da avere un riscontro immediato del suo funzionamento e delle sue criticità.

Procedendo nel porre le domande ho subito individuato due problemi:

- alla domanda "Che ore sono?" l'output segnalava un errore;

- la struttura WHILE non funzionava.

Ho perciò proseguito analizzando innanzitutto la sintassi dei singoli passaggi, per capire se ci fossero errori che, in effetti, ho trovato e di cui parlerò in seguito nel dettaglio.(vedi ERRORI)

Sono partita dal considerare gli errori di sintassi in quanto sono i più comuni e semplici da commettere: i classici errori di battitura o di distrazione.

Un altro importante dettaglio che ho voluto approfondire è il metodo LOWER, applicato solo ad un passaggio e non agli altri, senza un motivo evidente.(vedi ERRORI)

Successivamente ho cercato di fare delle considerazioni:

- A quante e a quali domande il programma è in grado di rispondere?

- Ci sono casi in cui il programma potrebbe crashare che non sono stati considerati?

- Ci sono casi specifici di domanda/risposta che potrebbero presentarsi?(vedi CASISTICHE NON STANDARD)

3) ERRORI

Primo errore di sintassi:

```
elif comando == "Che ore sono?":
    ora_attuale = datetime.datetime.now().time()
    risposta = "L'ora attuale è " + ora_attuale.strftime("%H↵%M")
```

Possiamo individuarlo nei caratteri sottolineati; il modo corretto di scriverlo è il seguente:

```
elif comando == "Che ore sono?":
    ora_attuale = datetime.date.time.now().time()
    risposta = "L'ora attuale è " + ora_attuale.strftime("%H↵%M")
```

Secondo errore di sintassi:

```
while True__
  comando_utente = input("Cosa vuoi sapere? ")
```

```
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
```

Anche qui ho evidenziato (sottolieandolo) il passaggio errato. Di seguito riporto la correzione:

```
while True:
    comando_utente = input("Cosa vuoi sapere? ")
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
```

Terzo errore logico:

Il metodo LOWER è utilizzato per convertire tutti i caratteri in una stringa in lettere minuscole. E' utile perchè permette di gestire l'input: se l'utente digita dei caratteri in masiuscolo, il metodo LOWER permetterà al programma di convertirli in minuscolo ed eseguirli di conseguenza.

```
while True:
    comando_utente = input("Cosa vuoi sapere? ")
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
```

In questo caso specifico, anche se l'utente digita la parola ESCI completamente in maiuscolo o solo parzialmente, il programma non avrà problemi ad elaborarla.

Non è perciò funzionale applicarlo soltanto in un determinato passaggio, ma è certamente meglio utilizzarlo in tutti i passaggi di input. (vedi SOLUZIONE)

4) CASISTICHE NON STANDARD

- A quante e a quali domande il programma è in grado di rispondere?

Per ogni ciclo IF/ELIF è stato impostato un solo modo di porre la domanda. **Questo rende il codice troppo rigido**:

```
elif comando == "Che ore sono?":
    ora_attuale = datetime.datetime.now().time()
    risposta = "L'ora attuale è " + ora_attuale.strftime("%H↵%M")
```

Se l'utente digita "Che ora è?" al posto di "Che ore sono?" il programma non è in grado di rispondere.

- Ci sono casi in cui il programma potrebbe crashare che non sono stati considerati?

E' sempre consigliato l'utilizzo del ciclo TRY - EXCEPT per evitare che il programma vada in CRASH nel momento in cui non riesce ad elaborare un dato qualsiasi, non considerato all'interno del codice; in questo caso, per esempio, potrebbe non reperire correttamente la data o l'ora e, di conseguenza, smettere di funzionare.

- Ci sono casi specifici di domanda/risposta che potrebbero presentarsi?

Se per sbaglio l'utente dovesse digitare lo spazio, il programma dovrebbe essere in grado di elaborare una stringa vuota (priva di caratteri testuali). In questo caso è sufficiente impostare una risposta al caso specifico:

```
if not comando_utente:
  print("Inserimento non valido. Riprovare"
  continue
```

Vedremo inserito questo passaggio all'interno del codice completo, in SOLUZIONE.

5) SOLUZIONE

Possiamo riassumere le modifiche effettuate nel seguente elenco:

-correzione errori di sintassi e logici

-codice meno rigido

-inserimento casistica stringa vuota

-inserimento ciclo TRY - EXCEPT per gestire le eventuali eccezioni.

Queste azioni hanno reso il codice sicuramente più efficiente e meno problematico rispetto a quello originale.

Di seguito riporto il codice completo di correzioni:

```
import datetime

def assistente_virtuale(comando):
    try:
        if comando.lower() in ["mi dici la data di oggi?", "che giorno è oggi?", "oggi che giorno è?", "che data è oggi?"]:
            oggi = datetime.date.today()
            risposta = "oggi è " + oggi.strftime("%d/%m/%Y")
        elif comando.lower() in ["che ore sono?", "che ora è?", "mi dici l'ora?", "sai che ora è?"]:
            ora_attuale = datetime.datetime.now().time()
            risposta = "Sono le " + ora_attuale.strftime("%H:%M")
        elif comando.lower() in ["come ti chiami?", "chi sei?", "qual è il tuo nome?"]:
            risposta = "Sono il tuo Assistente Virtuale"
        else:
            risposta = "Non ho capito la tua domanda."
    except Exception as e:
            risposta = "Si è verificato un errore!"

    return risposta

while True:
    comando_utente = input("Cosa vuoi sapere? ")
    if not comando_utente:
```

```
        print("Inserimento non valido. Riprova!")
        continue
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
    else:
        print(assistente_virtuale(comando_utente))
```

_