Progetto S2/L5

Traccia:

Per agire come un Hacker bisogna capire come pensare fuori dagli schemi. L'esercizio di oggi ha lo scopo di allenare l'osservazione critica. Dato il codice si richiede allo studente di:

- Capire cosa fa il programma senza eseguirlo.
- Individuare dal 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.

Codice Fornito:

```
import datetime

def assistente_virtuale(comando):
    if comando == "Qual è la data di oggi?":
        oggi = datetime.date.today()
        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))
```

Svolgimento:

Introduzione

In questa analisi, ho esaminato un semplice programma Python, forse sviluppato per funzionare come un assistente virtuale. Da quanto ho capito il fine del programma è rispondere a specifiche domande dell'utente, come la data odierna, l'ora corrente, e il nome dell'assistente.

Ho pensato di valutare secondo i seguenti criteri:

- Il funzionamento del programma
- I limiti legati alla gestione dei comandi
- Le potenziali vulnerabilità.

Alla fine, ho proposto una soluzione che secondo il mio parere andrà a migliorare l'usabilità e la sicurezza del programma, rendendolo più robusto e flessibile.

Analisi del codice

Il codice fornito genera un “assistente virtuale” che risponde a tre comandi specifici:

- "Qual è la data di oggi?"
- "Che ore sono?"
- "Come ti chiami?"

Se il comando non è programmato, risponde con "Non ho capito la tua domanda."
Il programma continua a funzionare fino a quando l'utente non digita "esci".

Identificazione dei problemi

1- Input non sensibile a maiuscole/minuscole

- Il programma non gestisce variazioni nei comandi, come minuscole o variazioni di parole.
    - Possibile soluzione: convertire il comando in minuscolo e utilizzare parole chiave per il riconoscimento, rendendo il programma più reattivo.

2- Poco chiaro per comandi non riconosciuti

- Il messaggio di errore ("Non ho capito la tua domanda.") non fornisce suggerimenti all'utente.
    - Possibile soluzione: Suggerire i comandi corretti.

2- Assenza di gestione delle eccezioni

- Nessuna gestione degli errori è presente, in caso di errori imprevisti (come problemi con il modulo datetime) possono bloccare il programma.
    - Possibile soluzione: Aggiungere blocco try/try-except per evitare che il programma si blocchi.

Conclusione analisi

Dall'analisi effettuata, ho identificato alcune aree da igliorare per ottimizzare il codice e rendere l'assistente virtuale più resiliente.
Le modifiche suggerite, tra cui l'uso di parole chiave per un riconoscimento dei comandi più flessibile e l'implementazione di blocchi di gestione delle eccezioni, contribuiscono a migliorare la capacità del programma di gestire input non standard e di evitare interruzioni indesiderate.

Soluzione alternativa: Uso di un dizionario di comandi

Possiamo utilizzare un dizionario per i comandi con funzioni specifiche, questo dovrebbe rendere il codice più reattivo.

```
import datetime


def data_oggi():
    oggi = datetime.date.today()
    return "La data di oggi è " + oggi.strftime("%d/%m/%Y")


def ora_attuale():
    ora = datetime.datetime.now().time()
    return "L'ora attuale è " + ora.strftime("%H:%M")


def nome_assistente():
    return "Mi chiamo Assistente Virtuale"
```

```
def assistente_virtuale(comando):
    #dizionario di comandi e funzioni
    comandi = {
        "data": data_oggi,
        "ore": ora_attuale,
        "nome": nome_assistente,
        "chiami": nome_assistente
    }

    #verifica se una delle parole chiave è nel comando
    for chiave, funzione in comandi.items():
        if chiave in comando:
            return funzione()  #esegue la funzione associata alla parola chiave

    #messaggio di errore se il comando non è riconosciuto
    return ("Comando non riconosciuto. "
            "Prova con: 'Qual è la data di oggi?', 'Che ore sono?', o 'Come ti chiami?'")

while True:
    try:
        comando_utente = input("Cosa vuoi sapere tra le opzioni sotto elencate? (digita 'esci' per uscire): \n"
                               "Qual è la data di oggi?, 'Che ore sono?', o 'Come ti chiami?'")
        if comando_utente.lower() == "esci":
            print("Arrivederci!")
            break
```

```
        else:
            print(assistente_virtuale(comando_utente.lower()))
    except Exception as e:
        print(f"Si è verificato un errore: {e}")
```

Vantaggi

- Aggiungere nuovi comandi è più semplice. Basta aggiungere una nuova funzione e una nuova chiave nel dizionario.
- Il codice è più leggibile e separato in funzioni indipendenti.
- Possiamo facilmente estendere il dizionario con nuove parole chiave o funzioni.

Conclusione Proposta

Ho effettuato questa proposta per mantenere il codice semplice, diciamo che la scelta migliore dipende dal contesto e dall'eventuale espansione del programma.
L'uso del dizionario di comandi è probabilmente la scelta migliore.
Se si pensa che il programma deve crescere in complessità, allora si può provare l'uso di una classe o di espressioni regolari.