S2.E5

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.

Esecuzione:

L'esercizio di oggi sarà svolto utilizzando Kali Linux in VR. Iniziamo scrivendo il codice e segnaliamo eventuali errori con un commento (#err1, err2, ecc). Dal codice ci accorgiamo subito che il programma riguarda un assistente virtuale, il quale risponderà a quattro comandi, tre dei quali sono domande: "Qual è la data di oggi?", "Che ore sono?", "Come ti chiami?" e, infine, il comando "esci".

```
1 import datetime
2 def assistente_virtuale(comando):
3     if comando == "Qual è la data di oggi?":
4         oggi = datetime.datetoday() #err1
5         risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
6     elif comando == "Che ore sono?":
7         ora_attuale = datetime.datetime.now().time() #err2
8         risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
9     elif comando == "Come ti chiami?":
10        risposta = "Mi chiamo Assistente Virtuale"
11    else:
12        risposta = "Non ho capito la tua domanda."
13    return risposta
14 while True #err3
15    comando_utente = input("Cosa vuoi sapere? ")
16    if comando_utente.lower() == "esci":
17        print("Arrivederci!")
18        break
19    else:
20        print(assistente_virtuale(comando_utente))
21
```

Dopo aver esaminato il codice, possiamo confermare la presenza di tre errori:

- **#err1**: Il primo errore riguarda la funzione, che è scritta in modo errato. Viene utilizzata datatime.datatoday(), una funzione inesistente.
- **#err2**: Il secondo errore deriva dall'utilizzo di .time(), che è un'istruzione inutile.
- **#err3**: Il terzo errore riguarda l'assenza di un simbolo, ovvero i due punti dopo la funzione true, il che impedisce il corretto funzionamento del codice.

Avendo individuato gli errori correggiamo il codice :

```
1 import datetime
2 def assistente_virtuale(comando):
3     if comando == "Qual è la data di oggi?":
4         oggi = datetime.datetime.now()
5         risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
6     elif comando == "Che ore sono?":
7         ora_attuale = datetime.datetime.now()
8         risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
9     elif comando == "Come ti chiami?":
10        risposta = "Mi chiamo Assistente Virtuale"
11    else:
12        risposta = "Non ho capito la tua domanda."
13    return risposta
14 while True:
15    comando_utente = input("Cosa vuoi sapere? ")
16    if comando_utente.lower() == "esci":
17        print("Arrivederci!")
18        break
19    else:
20        print(assistente_virtuale(comando_utente))
21
```

Correggiamo il primo errore scrivendo la funzione corretta, ovvero: datetime.datetime.now().

Successivamente, procediamo con la correzione del secondo errore, eliminando .time() poiché è un'informazione superflua.

Infine, correggiamo il terzo errore aggiungendo i due punti dopo la funzione true.

Per utilizzare il nostro codice su Linux, salveremo il file con l'estensione ".py" (python).

Oltre tutto ciò individuiamo diverse casistiche non standard che non vengono gestite e che potrebbero causare problemi o comportamenti imprevisti tra cui:

I comandi con variazione tra minuscole e maiuscole non verranno riconosciuti, ad esempio CHE ORE SONO? Ci darà come risposta “Non ho capito la domada”, ciò

avviene per tutti i comandi ad eccezione di “esci” essendo stato implementato con la funzione comando_utente.lower(), per far si che questa evenienza non si verifichi, bisogna implementare agli altri comandi con la medesima funzione.

Un altra casistica non gestita riguarda il ciclo while true, che continua a chiedere input all’utente, per risolvere questa problematica bisognerebbe aggiungere una funzione per interrompere il loop in caso di inattivtà.

Aggiungendo controlli per queste casistiche non standard, il programma riuscirebbe a gestire input inaspettati.