Analisi Errori - Assistente Virtuale

Informazioni Generali

Studente: Rocco Paolo Caccamo
Corso: Cybersecurity Specialist
Data: 05/12/2025
Esercizio: Analisi del codice senza eseguirlo (Hacker Mindset)

Descrizione del Programma

L'obiettivo del programma è creare un **Assistente Virtuale** in grado di:

- Comunicare la data di oggi
- Comunicare l'ora attuale
- Comunicare il proprio nome

Il programma utilizza un ciclo infinito che continua fino a quando l'utente non digita il comando "esci".

Strutture di controllo principali: `if`, `elif`, `else`, ciclo `while`, funzioni

---

SEZIONE 1: ERRORI DI SINTASSI

Errore di Sintassi n.1 - Riga 13

Istruzione errata:
oggi = datetime.datetoday()

Descrizione: Il metodo `datetoday()` non esiste nel modulo `datetime`.

Perché è un errore di sintassi: Python ricerca il metodo `datetoday()` direttamente nel modulo `datetime`, ma questo metodo non è definito. Il programma solleverà un `AttributeError` durante l'esecuzione.

Correzione proposta: Usare il metodo corretto `datetime.date.today()` con un punto tra `datetime`, `date` e `today()`.

oggi = datetime.date.today()

Errore di Sintassi n.2 - Riga 11

Istruzione errata:
Definizione della funzione assistente_virtuale() dopo il suo utilizzo nel ciclo while

Descrizione: La funzione `assistente_virtuale()` viene richiamata alla riga 9 del ciclo `while`, ma viene definita solo alla riga 11.

Perché è un errore di sintassi: Python legge il codice dall'alto verso il basso. Quando incontra la chiamata `assistente_virtuale(comando_utente)` alla riga 9, non ha ancora "visto" la definizione della funzione, quindi genera un `NameError: name 'assistente_virtuale' is not defined`.

Correzione proposta: Spostare la definizione della funzione (righe 11-22) **PRIMA** del ciclo `while` (righe 3-9). L'ordine deve essere:

1. `import datetime`
2. (riga vuota)
3. `def assistente_virtuale(comando):` ← DEFINISCI QUI
4. corpo della funzione
5. (riga vuota)
6. `while True:` ← POI FAI IL CICLO

Errore di Sintassi n.3 - Righe 3-9

Istruzione errata:
Rientro (indentazione) non coerente nel blocco while

Descrizione: Nel codice originale è visibile un carattere blu/punto alla riga 7 dopo `break`, e la riga 9 con `print(assistente_virtuale(comando_utente))` ha un'indentazione non allineata all'`else:`.

Perché è un errore di sintassi: Python è molto sensibile agli spazi e alle tabulazioni. Se il rientro non è coerente, il parser non riesce a identificare correttamente i blocchi di codice (`if`, `else`, `while`) e genera un `IndentationError`.

Correzione proposta: Verificare che tutto il codice nel blocco `while` abbia un rientro coerente (4 spazi o 1 tab per ogni livello). In particolare, il `break` e il `print()` devono stare sullo stesso livello di indentazione rispetto ai loro blocchi `if` ed `else`.

SEZIONE 2: ERRORI LOGICI E CASI NON GESTITI

Errore Logico n.1 - Righe 5, 12, 15, 18 (Sensibilità al Maiuscolo/Minuscolo)

Descrizione: Il programma confronta gli input dell'utente con stringhe esatte che contengono maiuscole:

- `if comando_utente == "esci":`
- `if comando == "Qual è la data di oggi?":`
- `elif comando == "Che ore sono?":`
- `elif comando == "Come ti chiami?":`

Cosa fa attualmente il programma: Se l'utente digita "ESCI", "Esci", "QUAL È LA DATA DI OGGI?", "qual è la data di oggi?", il programma **non riconosce il comando** e risponde sempre con "Non ho capito la tua domanda." oppure continua il ciclo senza uscire.

Esempi di comportamento errato:

- Input: `ESCI` → Risposta: "Non ho capito la tua domanda." (non esce dal programma)

- Input: `Qual è la data di oggi?` → Risposta: "Non ho capito la tua domanda." (maiuscolo iniziale non riconosciuto)
- Input: `qual è la data di oggi?` → Risposta: "Non ho capito la tua domanda." (tutto minuscolo non riconosciuto)

Cosa dovrebbe fare invece: Indipendentemente da come l'utente digita il comando (maiuscolo, minuscolo, misto), il programma dovrebbe riconoscerlo e rispondere correttamente.

Correzione proposta: Convertire sempre l'input a minuscolo usando il metodo `.lower()` prima di fare il confronto:

```
comando_utente = input("Cosa vuoi sapere? ").lower()
```

Quindi nel ciclo `while`:
```
if comando_utente == "esci":
```

E nella funzione, confrontare sempre stringhe in minuscolo:
```
if comando == "qual è la data di oggi?":
...
elif comando == "che ore sono?":
...
elif comando == "come ti chiami?":
...
```

Errore Logico n.2 - Righe 4, 12, 15, 18 (Spazi Extra negli Input)

Descrizione: Il programma confronta stringhe esatte senza rimuovere gli spazi iniziali e finali.

Cosa fa attualmente il programma: Se l'utente digita `" esci "` (con spazi prima e dopo), oppure `" Qual è la data di oggi? "`, il programma **non riconosce il comando** perché la stringa contiene caratteri di spazio aggiuntivi che non corrispondono esattamente.

Esempi di comportamento errato:

- Input: `esci` (con uno spazio prima e dopo) → Risposta: "Non ho capito la tua domanda." (non esce)
- Input: `qual è la data di oggi?` → Risposta: "Non ho capito la tua domanda."

Cosa dovrebbe fare invece: Ignorare gli spazi iniziali e finali e riconoscere comunque il comando.

Correzione proposta: Usare il metodo `.strip()` per rimuovere spazi iniziali e finali:

```
comando_utente = input("Cosa vuoi sapere? ").lower().strip()
```

Combinando `.lower()` e `.strip()` si risolvono sia il problema del Caps Lock che degli spazi extra.

Caso Non Gestito n.1 - Input Vuoto

Descrizione del caso: L'utente preme Invio senza digitare nulla.

Effetto atteso sul programma: Il programma dovrebbe gestire l'input vuoto in modo appropriato (ad esempio, chiedere di nuovo la domanda oppure mostrare un messaggio di errore).

Perché non gestito dal codice attuale: Il programma non ha un controllo specifico per input vuoti. Se l'utente digita solo Invio, la variabile `comando_utente` avrà il valore `""` (stringa vuota), che non corrisponde a nessuno dei comandi previsti e finisce nell'`else`, restituendo "Non ho capito la tua domanda."

Soluzione consigliata: Aggiungere un controllo:

```
if comando_utente == "":
print("Per favore, digita un comando valido.")
continue
```

Caso Non Gestito n.2 - Comandi Simili ma Non Identici

Descrizione del caso: L'utente digita comandi leggermente diversi da quelli previsti, come:

- "Che ore sono adesso?" (aggiunge "adesso")
- "Qual è la data di oggi allora?" (aggiunge "allora")
- "Come ti chiami tu?" (aggiunge "tu")
- "Dimmi la data" (formulazione diversa)

Effetto atteso sul programma: Idealmente, il programma dovrebbe riconoscere l'intenzione anche con variazioni minori.

Perché non gestito dal codice attuale: Il programma confronta stringhe esatte senza riconoscere variazioni. Anche con `.lower()` e `.strip()`, qualsiasi piccola modifica farà sì che il comando finisca nell'`else`.

Soluzione consigliata: Usare metodi di matching più sofisticati come `in` (verificare se una parte della stringa è contenuta) oppure usare espressioni regolari. Esempio:

```
if "data" in comando and "oggi" in comando:
# Rispondi con la data
elif "ore" in comando or "orario" in comando:
# Rispondi con l'ora
elif "nome" in comando or "chiami" in comando:
# Rispondi con il nome
```

Caso Non Gestito n.3 - Caratteri Speciali e Punteggiatura Variabile

Descrizione del caso: L'utente digita comandi con punteggiatura diversa:

- "qual è la data di oggi" (senza punto interrogativo)
- "qual è la data di oggi ?" (spazio prima del punto interrogativo)
- "Qual E la data di oggi?" (accento diverso su "è")

Effetto atteso sul programma: Il programma dovrebbe accettare il comando indipendentemente dalla punteggiatura.

Perché non gestito dal codice attuale: Il programma confronta le stringhe esatte, inclusa la punteggiatura. Se manca il punto interrogativo o se è posizionato diversamente, il confronto fallisce.

Soluzione consigliata: Rimuovere la punteggiatura prima del confronto usando `.replace()` o `string.punctuation`:

```
import string
comando_pulito = comando_utente.translate(str.maketrans('', '', string.punctuation))
```

Oppure più semplicemente:

```
comando_pulito = comando_utente.replace("?", "").replace("!", "").replace(".", "")
```

SEZIONE 3: TABELLA RIEPILOGATIVA DEGLI ERRORI

Riga	Tipo	Errore	Gravità	Soluzione
13	Sintassi	datetime.datetoday() non esiste	Critico	Usare datetime.date.today()
11	Sintassi	Funzione definita dopo l'uso	Critico	Spostare prima del ciclo while
3-9	Sintassi	Indentazione non coerente	Critico	Verificare spazi e tabulazioni
5, 12, 15	Logico	Case-sensitive negli input	Alto	Usare .lower() su input
4, 12, 15	Logico	Spazi extra negli input	Medio	Usare .strip() su input

Table 1: Riepilogo degli errori di sintassi e logica

SEZIONE 4: CODICE CORRETTO COMPLETO

```
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? ").lower().strip()
if comando_utente == "esci":
print("Arrivederci!")
break
else:
print(assistente_virtuale(comando_utente))
```

SEZIONE 5: ESEMPI DI TEST COMPARATIVI

Input Utente	Codice Originale	Codice Corretto
esci	✅ Esce	✅ Esce
ESCI	❌ Non riconosce	✅ Esce
esci (con spazi)	❌ Non riconosce	✅ Esce
Qual è la data di oggi?	✅ Mostra data	✅ Mostra data
qual è la data di oggi?	❌ Non riconosce	✅ Mostra data
QUAL È LA DATA DI OGGI?	❌ Non riconosce	✅ Mostra data
qual è la data?	❌ Non riconosce	✅ Mostra data
Che ore sono?	✅ Mostra ora	✅ Mostra ora
che ore sono?	❌ Non riconosce	✅ Mostra ora
Come ti chiami?	✅ Risponde nome	✅ Risponde nome

Table 2: Test comparativi tra codice originale e corretto

CONCLUSIONI E RACCOMANDAZIONI

1. **Errori critici** (Sintassi): I tre errori di sintassi impediscono al programma di funzionare. Devono essere corretti prima di qualsiasi test.
2. **Errori logici** (Robustezza): L'uso di `.lower()` e `.strip()` rende il programma molto più robusto e user-friendly.
3. **Casi non gestiti**: Con `.lower()` e `.strip()`, il programma gestisce correttamente la sensibilità al maiuscolo e gli spazi extra, rendendolo robusto per l'uso basico.