

Esercitazione 2° Settimana

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

1. Capire cosa fa il programma

In sintesi, il codice crea un "assistente virtuale" a cui si possono porre domande.

Come funziona?

- Possiamo notare, in primis, la definizione di una funzione `assistente_virtuale`, che richiede un parametro comando. Dentro la funzione è presente un blocco `if-elif-else` in cui, se il valore di comando corrisponde a una delle opzioni contemplate dal codice, viene restituita una determinata risposta. In caso contrario, viene restituita una risposta generica: "Non ho capito la tua domanda", così da dare comunque un output all'utente. La funzione restituisce infine la variabile `risposta`, che contiene l'output corrispondente al valore del parametro comando.
- Dopo la funzione troviamo un ciclo `while True` (loop infinito), in cui viene prima richiesto un input all'utente. Se l'utente digita "esci" (case insensitive), il programma interrompe il loop tramite un `break`. Altrimenti, il valore inserito dall'utente viene passato come parametro alla funzione `assistente_virtuale`, che restituisce la risposta dell'assistente virtuale e riprende il ciclo richiedendo nuovamente un input.

2. Casistiche non standard

Il codice presenta alcune casistiche non standard o punti in cui poteva essere scritto in modo migliore:

- **2.a** Il parametro comando dovrebbe essere gestito come case insensitive, così da rendere meno severo l'inserimento dell'utente.
- **2.b** Sarebbe utile ignorare il carattere "?", poiché tutte le domande sono espresse in modo interrogativo e il punto interrogativo risulta ridondante. Questo semplificherebbe l'inserimento per l'utente.

3. Errori di sintassi e logici

Errori di sintassi

- **3.a** Nella riga 4 del codice, l'assegnazione della variabile oggi utilizza in modo errato il modulo `datetime`. `datetime` possiede solo i metodi `datetime` e `date`, quindi il metodo `datetime.datetime.today()` è sbagliato poiché non esiste.
- **3.b** Nel ciclo `while True` manca il simbolo `:` dopo `True`, necessario per delimitare il blocco logico.

Errori logici

- **3.c** L'utente è poco guidato al momento dell'inserimento dell'input. Sarebbe utile fornire istruzioni chiare su cosa l'assistente virtuale può fare e su come uscire dal programma.
- **3.d** Il codice non è ottimizzato e poco scalabile. L'utilizzo di un ciclo if-elif-else funziona solo se la funzione gestisce poche domande, ma diventa inefficiente e difficile da mantenere man mano che aumentano le domande.

4. Proposte di soluzioni

2.a Rendere il parametro case insensitive

Per rendere il parametro case insensitive, si può utilizzare il metodo `.lower()`. Ad esempio:

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

In questo modo, il confronto delle stringhe sarà indipendente dalle maiuscole/minuscole.

2.b Ignorare i caratteri speciali

Per ignorare caratteri come `?`, si può utilizzare la libreria `re` per rimuovere tutti i caratteri non alfanumerici:

```
import re
comando_utente = re.sub(r"[^\w\s]","", comando_utente)
```

3.a Correggere il metodo di datetime

Il comando corretto per ottenere la data odierna è:

```
oggi = datetime.date.today()
```

3.b Correggere il ciclo while

Basta aggiungere `i : dopo True`:

```
while True:
```

3.c Guidare meglio l'utente

Si può aggiungere un messaggio iniziale che spiega come utilizzare l'assistente virtuale:

```
print("Benvenuto! Ecco come utilizzare l'assistente virtuale:\n" "- Scrivi la tua domanda per ricevere una risposta.\n" "- Digita 'comandi' per vedere l'elenco delle domande disponibili.\n" "- Digita 'esci' per chiudere il programma.")
```

creo quindi la lista delle domande disponibili

```
lista_domande = ["Qual è la data di oggi?", "Che ore sono?", "Come ti chiami?"]
```

```
while True:
```

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

```
    if comando_utente == "comandi":
        print(f"Domande disponibili: {lista_domande}")
```

```
    elif comando_utente == "esci":
```

```
break
```

```
else:
```

```
    print(assistente_virtuale(comando))
```

All'avvio del programma verrà mostrato un messaggio che spiega come utilizzare l'assistente virtuale. Inoltre, ho aggiunto la possibilità di visualizzare a schermo la lista delle domande a cui l'assistente virtuale è stato programmato per rispondere.

3.d Ottimizzare il codice per scalabilità

Per rendere il codice più scalabile, si possono creare funzioni per ogni comando e un dizionario per far corrispondere le domande alle funzioni.

Iniziamo quindi scrivendo le funzioni per ogni risposta che l'assistente virtuale dà

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

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

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

def comando_non_riconosciuto():
    return "Non ho capito la domanda."
```

Creiamo poi il dizionario che avrà come chiave la Domanda e come valore la funzione che chiamata darà la risposta

```
comandi_disponibili = { "Qual è la data di oggi?": data_di_oggi, "Che ore sono?": ora_attuale,
    "Come ti chiami?": nome_assistente }
```

e creiamo poi la funzione `assistente_virtuale` che verrà definita con il parametro `comando` e quando chiamata cercherà nel dizionario creato (`comandi_disponibili`) il valore corrispondente al comando se non presente userà di default la funzione `comando_non_riconosciuto`

```
def assistente_virtuale(comando):
    return comandi_disponibili.get(comando, comando_non_riconosciuto)()
```

In questo modo, il codice risulta più ordinato e veloce, rendendo molto più semplice aggiungere nuove risposte a cui l'assistente virtuale può rispondere, migliorandone così la scalabilità.