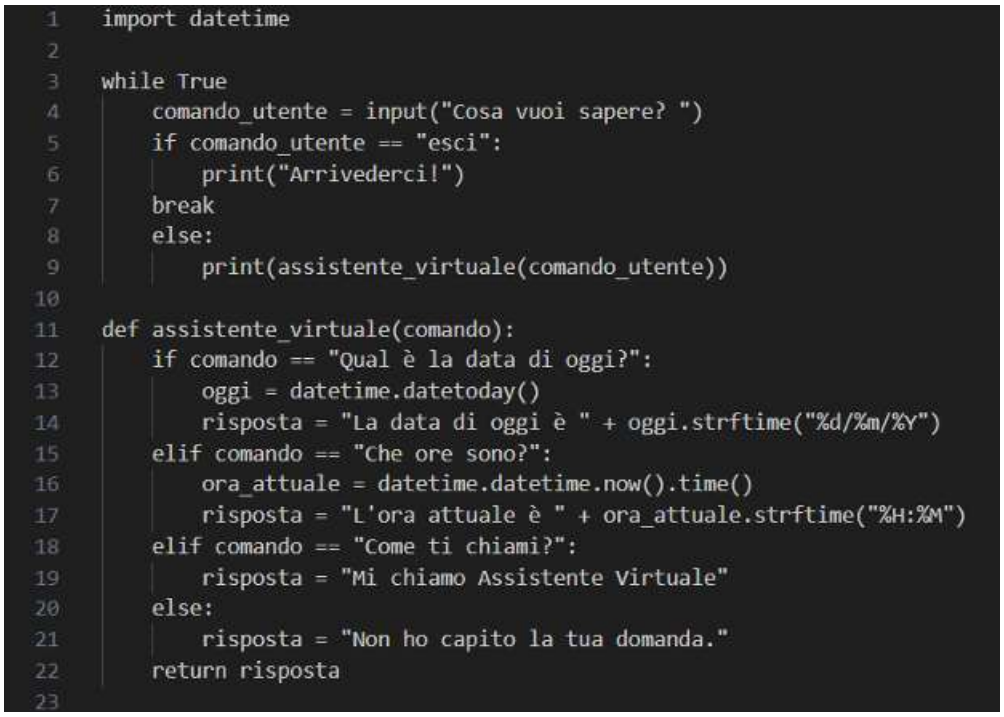
PROGETTO S2/L5 – LEONARDO TAKESHI CHIAVERINI

1 INTRODUZIONE

In questa relazione verrà analizzato un progetto in cui il programma oggetto di esame è un semplice assistente virtuale testuale. Questo programma è stato sviluppato con l'obiettivo di rispondere a determinate domande dell'utente e di terminare l'esecuzione su richiesta.

Nel corso di questa analisi, verranno evidenziati gli errori individuati e le correzioni apportate, con l'obiettivo di fornire un report chiaro e completo.



```
1   import datetime
2
3   while True
4       comando_utente = input("Cosa vuoi sapere? ")
5       if comando_utente == "esci":
6           print("Arrivederci!")
7       break
8       else:
9           print(assistente_virtuale(comando_utente))
10
11  def assistente_virtuale(comando):
12      if comando == "Qual è la data di oggi?":
13          oggi = datetime.datetoday()
14          risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
15      elif comando == "Che ore sono?":
16          ora_attuale = datetime.datetime.now().time()
17          risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
18      elif comando == "Come ti chiami?":
19          risposta = "Mi chiamo Assistente Virtuale"
20      else:
21          risposta = "Non ho capito la tua domanda."
22      return risposta
23
```

Sorgente progetto

2 ERRORI DI SINTASSI

Nel codice originario sono stati individuati diversi errori di sintassi e di struttura che compromettono il corretto funzionamento del programma. Di seguito si riportano i principali errori accompagnati da una breve analisi e proposta di soluzione, con il relativo estratto di codice e numerazione delle righe del sorgente fornito nel progetto.

2.1 Mancanza dei due punti alla fine del while (riga 3)

Codice originale:

```
3   while True
```

Spiegazione: In python, le istruzioni di controllo devono obbligatoriamente terminare con i due punti (:) per introdurre il blocco di codice associato.

Correzione: Con l'aggiunta dei due punti il blocco viene riconosciuto correttamente come corpo del ciclo while.

```
3   while True:
```

2.2 Posizionamento errato del break (riga 7)

Codice originale:

```
5       if comando_utente == "esci":
6           print("Arrivederci!")
7       break
```

Spiegazione: Il *break* (riga 7) è allineato con l'*if* (riga 5) invece di essere indentato all'interno del blocco; in questo modo verrebbe eseguito indipendentemente dal valore di *comando_utente*, causando l'uscita immediata dal ciclo alla prima iterazione.

Correzione: Nel seguente modo il *break* è correttamente indentato all'interno del blocco *if* ed è eseguito solo quando l'utente inserisce il comando di uscita.

```
5       if comando_utente == "esci":
6           print("Arrivederci!")
7           break
```

2.3 Struttura non valida dell'*if/else* (righe 5-9)

Codice originale:

```
5       if comando_utente == "esci":
6           print("Arrivederci!")
7       break
8       else:
9           print(assistente_virtuale(comando_utente))
```

Spiegazione: La criticità in oggetto è una diretta conseguenza della precedente. La presenza del *break* fuori dal blocco sotto *if* viola la sintassi di Python in merito al

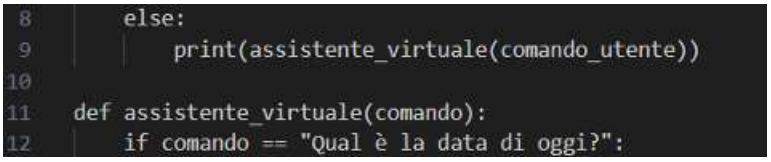
costrutto *if...else*; *if* e *else* devono essere consecutivi e non possono esserci istruzioni "sparse" tra i due. La presenza del *break* allineato con l'*if* interrompe questa struttura e genera un errore di sintassi.

Correzione: Pertanto con la correzione della precedente osservazione, la struttura *if/else* rispetta la sintassi prevista dal linguaggio, permettendo a *else* di seguire direttamente il blocco *if*.

```
5    if comando_utente == "esci":
6        print("Arrivederci!")
7        break
8    else:
9        print(assistente_virtuale(comando_utente))
```

2.4 Utilizzo della funzione *assistente_virtuale* prima della sua definizione (righe 9-10 vs 11)

Codice originale:



Spiegazione: Nel sorgente analizzato, la funzione *assistente_virtuale* (riga 9) risulta definita dopo il ciclo while che la richiama (riga 11); tale struttura è scorretta dal punto di vista logico-organizzativo. Poiché il ciclo viene eseguito immediatamente all'avvio del programma, questa scelta comporta un potenziale errore di runtime (*NameError*).

Correzione: Per una struttura corretta, le funzioni dovrebbero essere definite prima del codice che le invoca. In questo modo, al momento della chiamata, la funzione risulta già definita e disponibile.

```
import datetime

def assistente_virtuale(comando):
    # corpo della funzione
    ...
while True:
    comando_utente = input("Cosa vuoi sapere? ")
    if comando_utente == "esci":
        print("Arrivederci!")
        break
```

```
else:
    print(assistente_virtuale(comando_utente))
```

2.5 Utilizzo di una funzione inesistente: *datetime.datetoday()* (riga 13)

Codice originale:

```
13          oggi = datetime.datetoday()
```

Spiegazione: Nel codice originale viene richiamato il metodo *datetime.datetoday()*, che non è presente nel modulo *datetime.* L'uso di un metodo non esistente genera un errore e impedisce al programma di proseguire l'esecuzione.

Correzione: Il modulo espone invece la classe *date*, la quale contiene il metodo corretto *today()*. L'istruzione corretta per ottenere la data corrente è pertanto:

oggi = datetime.date.today()

3 ERRORI LOGICO-FUNZIONALI E SCELTE POCO ROBUSTE

Qui non si tratta di errori di sintassi, ma di comportamenti che il codice non considera e che un hacker/analista dovrebbe notare.

3.1 Comandi riconosciuti solo se scritti identici (case sensitive, spazi, punteggiatura)

Per essere riconosciuti, i comandi devono essere scritti esattamente così:

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

Questo approccio rende l'interfaccia estremamente rigida.
Il programma riconosce il comando solo se la frase inserita dall'utente coincide esattamente con quella prevista (maiuscole, spazi e punteggiatura compresi).
Se ciò non accade allora il programma non capisce e va sempre nel ramo "Non ho capito la tua domanda." oppure non esce.

Questo è un **errore di progettazione dell'interfaccia** facendo risultare il programma troppo rigido.

Soluzione: Per cercare di mantenere lo stesso tipo di struttura e rendere l'assistente virtuale più robusto e vicino a un utilizzo reale, è opportuno normalizzare l'input prima di effettuare i confronti. La soluzione adottata consiste nell'applicare due trasformazioni principali alla stringa inserita dall'utente:

- ***strip()***: rimuove eventuali spazi bianchi iniziali e finali;
- ***lower()***: converte l'intera stringa in minuscolo, rendendo il confronto indipendente dall'uso di maiuscole/minuscole.

Nel ciclo principale, la lettura dell'input viene quindi modificata come segue:

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

A questo punto, tutti i controlli sul comando vengono effettuati utilizzando la versione normalizzata:

```
if comando_normalizzato == "esci":
        print("Arrivederci!")
        break
else:
        print(assistente_virtuale(comando_normalizzato))
```

In questo modo:

- il comando esci, con o senza spazi aggiuntivi e con qualsiasi combinazione di maiuscole/minuscole, viene riconosciuto correttamente;
- le domande principali possono essere scritte dall'utente anche in forme leggermente diverse sul piano delle maiuscole, senza compromettere il funzionamento del programma.

3.2 Mancata gestione degli input vuoti

Nel codice originale, se l'utente preme semplicemente Invio senza digitare alcun testo, il programma memorizza una stringa vuota ("") nella variabile *comando_utente*. Questa stringa viene poi passata direttamente alla funzione *assistente_virtuale*, la quale, non riconoscendola come un comando valido, restituisce il messaggio generico "Non ho capito la tua domanda.".
Sebbene questo comportamento non generi errori di esecuzione, rappresenta comunque un caso non gestito.

Soluzione: Prima di procedere all'elaborazione del comando, è opportuno verificare se l'utente abbia effettivamente digitato un contenuto significativo.
Per farlo, si utilizza il metodo *strip()*.

Il seguente codice permette di intercettare anche gli input composti solo da spazi:

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

    if not comando_utente.strip():
        print("Per favore, inserisci una domanda oppure 'esci'.")
        continue
```

3.3 Mancanza di un messaggio guida chiaro all'utente

Il programma chiede solo:

```
input("Cosa vuoi sapere? ")
```

Non vengono esplicitati quali comandi sono disponibili. Dal punto di vista "produttivo" per l'utente finale, è un errore di design e quindi il programma è poco usabile.

Soluzione: Se si vuole mantenere la stessa forma del sorgente proposto nel progetto è consigliabile stampare le azioni dispobili. All'avvio o ad ogni ciclo ci sarà:

```
print("Puoi chiedermi:\n")
print("Qual è la data di oggi?\n")
print("Che ore sono?\n")
print("Come ti chiami\n?")
print("Digita 'esci' per terminare.\n")
comando_utente = input("Cosa vuoi sapere? ")
```

Oppure un'ulteriore soluzione per rendere il programma più usabile, cercando di mantenere la struttura simile all'originale, consiste nell'introdurre un **menu numerato** delle opzioni disponibili.

In questo modo l'utente non è obbligato a digitare l'intera domanda in linguaggio naturale, ma può limitarsi a inserire solo il numero corrispondente alla funzionalità desiderata.

```
import datetime

def assistente_virtuale(comando):
```

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

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

    elif comando == "nome":
        return "Mi chiamo Assistente Virtuale"


while True:
    print("Seleziona un'opzione:")
    print("1 - Qual è la data di oggi?")
    print("2 - Che ore sono?")
    print("3 - Come ti chiami?")
    print("4 - Esci dal programma")

    scelta = input("Inserisci il numero dell'opzione: ").strip()

    if scelta == "1":
        print(assistente_virtuale("data"))
    elif scelta == "2":
        print(assistente_virtuale("ora"))
    elif scelta == "3":
        print(assistente_virtuale("nome"))
    elif scelta == "4":
        print("Arrivederci!")
        break
    else:
        print("Opzione non valida, riprova.")
```

4 CONCLUSIONI

L’analisi del codice originale ha evidenziato numerosi errori di sintassi e di struttura che impedivano al programma di funzionare correttamente. Gli interventi correttivi hanno riguardato sia aspetti formali del linguaggio Python (come l’allineamento dei blocchi, l’uso corretto dei due punti e delle virgolette, nonché la definizione delle funzioni prima del loro utilizzo), sia elementi logici relativi alla gestione dell’input e all’interazione con l’utente.