## *REPORT S2/L5*

Nell' esercizio di oggi è richiesto di analizzare criticamente il codice con lo scopo di capire cosa fa il programma senza eseguirlo, individuare dal codice sorgente le casistiche non standard che non vengono gestite (comportamenti non contemplati), individuare eventuali errori sintassi/logici e proporre una soluzione per ciascuno di questi.

```
import datetime

def assistente_virtuale(comando):

        if comando == "Qual è la data di oggi?":

                oggi = datetime.datetoday()

                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? ")

i       f comando_utente.lower() == "esci":

                print("Arrivederci!")

                break

        else:

                print(assistente_virtuale(comando_utente))
```

### ANALISI CODICE

Il codice preso in esame definisce una funzione ***assistente_virtuale()*** al quale viene passato un parametro in ingresso "comando" con lo scopo di gestire alcune domande. La funzione è in grado di rispondere a tre tipi di domande diverse riguardanti la data, l ' orario e il nome dell' assistente stesso, il tutto è possibile grazie al modulo daytime, importato ad inizio codice, il quale fornisce delle classi per la gestione della data e dell' ora. Nella parte successiva del codice all' interno di un loop (**while == true**) viene richiesto all' utente di porre una domanda alla macchina, nel caso in cui l' utente inserisca la stringa "esci" il programma termina ***break*** altrimenti viene richiamata la funzione definita precedentemente per rispondere alle richieste mediante una serie di ***if-elif***.

## ANALISI ERRORI

Dopo aver eseguito il codice andiamo a correggere gli eventuali errori indivituati:

- ***Errore1***: Errore di sintassi, non è presente ":" dopo il ***while***.

```
└─$ python progetto.py
  File "/home/kali/Desktop/python/progetto.py", line 16
    while True
              ^
SyntaxError: expected ':'
```

- ***Errore 2***: Un errore a runtime nel momento in cui si richiede la data, non esiste l' attributo datetoday del modulo datetime ma esiste **date.today()**.

```
    oggi = datetime.datetoday()
           ^^^^^^^^^^^^^^^^^^^
AttributeError: module 'datetime' has no attribute 'datetoday'
```

- ***Errore3***:Nell' eventualità in cui il parametro passato alla funzione non venga riconosciuto non c'è nessuna definizione della variabile ***risposta*** se non nel ***return***.
- ***Errore4***: Nonostante il metodo presente nel codice ***datetime.datetime.now().time()*** produca effettivamente un rissultato sarebbe sufficiente il metodo ***.datetime.now()***

```
from datetime import datetime

current_dateTime = datetime.now()

print(current_dateTime)
# 2022-09-20 10:27:21.240752
```

https://www.freecodecamp.org/news/python-datetime-now-how-to-get-todays-date-and-time/

## GESTIONE CASI PARTICOLARI

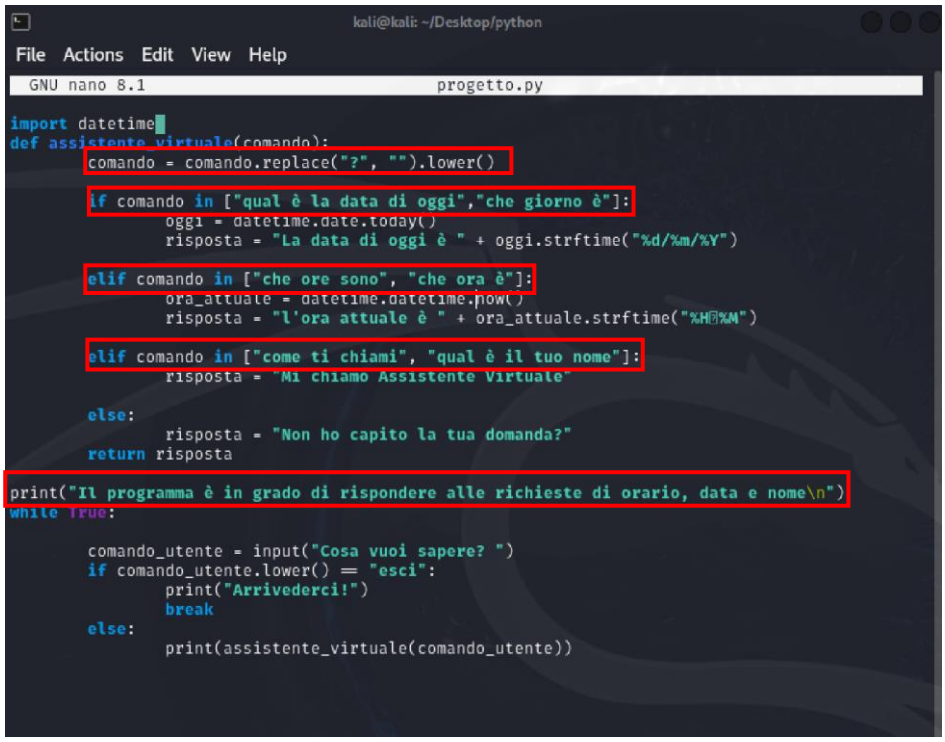
Andiamo a capire se sia possibile gestire meglio eventuali casi particolari. Analizzando il codice, da una prima occhiata è evidente che in primo luogo manca un' indicazione data all' utente su quali siano gli input che possano effettivamente generare una risposta che non sia **"non ho capito la domanda",** per risolvere questo problema si potrebbe immaginare di inserire per esempio ad inizio codice una ***print()*** che indichi a quali domande la funzione è in grado di rispondere. Un altro caso che puo essere gestito è quello di andare a convertire in ***lowercase*** tutte le stringhe date in input in maniera tale da garantire una maggiore possibilità di matchare le domande a cui la funzione sa rispondere per farlo all' inizio della funzione posso inserire la riga ***comando = comando.lower()*** facendo lo stesso con le stringhe nei controlli negli ***if.*** A questo si puo aggiungere quello di andare ad escludere l' eventualità di possibili errori di battitura fatti dall' utente in fase di inserimento dell' input, per farlo possiamo aggiungere una gestione piu flessibile degli input dell' utente, escludendo la punteggiatura. Aggiungo al comando visto precendentemente il metodo ***replace*** sostituendo il "?", ***.replace("?", "")***.

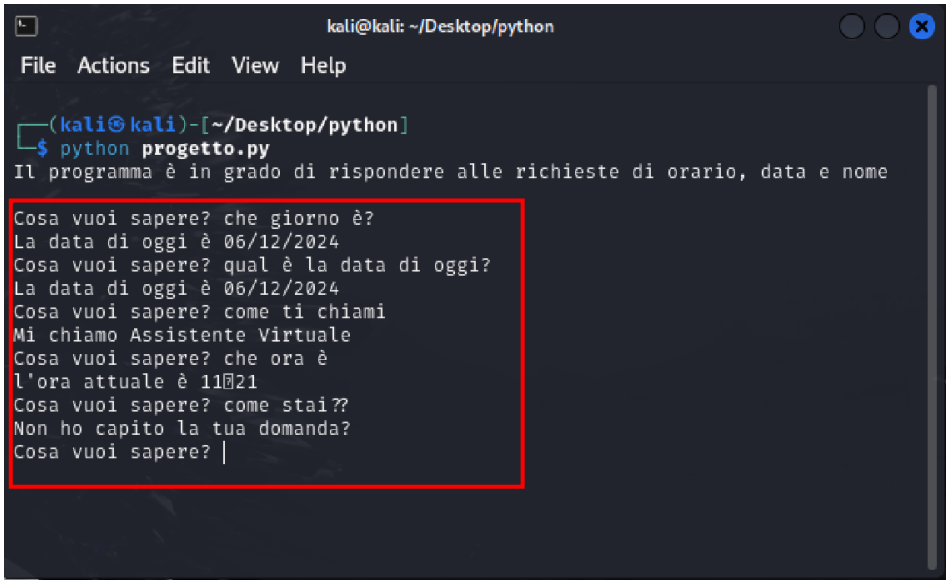
```
def assistente_virtuale(comando):
        comando = comando.replace("?", "").lower()
```

Posso poi aggiungere nell' if la riga: if ***comando in ["qual è la data di oggi", "che giorno è"],*** in modo da rendere il controllo sull' input piu versatile, ampliando la possibilità di matching dell' imput.

```
if comando in ["qual è la data di oggi","che giorno è"]:
        oggi = datetime.date.today()
        risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
```

Andiamo quinidi a vedere il nuovo codice corretto e provandolo in esecuzione con alcuni input per verificarne il funzionamento, in rosso sono evidenziate le aggiunte viste precedentemente.




### *ESERCIZIO BONUS*

Scrivi un programma Python per gestire una lista della spesa. Il programma deve permettere all'utente di:

1) Aggiungere un elemento alla lista.

2) Rimuovere un elemento dalla lista (se presente).

3) Visualizzare tutti gli elementi della lista ordinati in ordine alfabetico.

4) Salvare la lista su file. 5) Caricare una lista da file.

Il programma deve avere un menu che consente all'utente di scegliere le varie operazioni e deve terminare solo quando l'utente lo richiede.

## SOLUZIONE

Per prima cosa procederò con la definizione delle funzioni relative alle 5 operazioni che il programma deve svolgere sulle liste.

La prima funzione ***aggiungi_elemento(lista)*** prenderà come parametro la lista (inizializzata come lista vuota), procederà poi al controllo sull' elemento richiesto in input all' utente per assicurarsi che l' elemento inserito non sia vuoto, successivamente se l' operazione è valida allora si procede con l' aggiunta.

```
def aggiungi_elemento(lista):
        x = input("Aggiungi un elemento: ").strip()
        if x:
                lista.append(x)
        else:
                print("Non puo essere aggiunto alla lista")
```

La seconda funzione ***rimuovi_elemento(lista)*** prende come parametro la lista e dopo aver controllato che l' elemento sia presente nella lista (*if x in lista*) stessa andrà a rimuoverlo dalla stessa.

```
def rimuovi_elemento(lista):

        x = input("Qaule elemento vuoi rimuovere?").strip()
        if x in lista:
                lista.remove(x)
                print("Rimozione avvenuta")
                return lista
        else:
                print("Non è stato possibile effettuare la rimozione")
```

La terza funzione ***visualizza_lista(lista)*** semplimente procederà con la stampa della lista ordinata in ordine alfabetico mediante il metodo ***sort()*** a meno che questa non sia una lista vuota in quel caso avvertirà con una stringa predefinita.

```
def visualizza_lista(lista):
        if lista:
                lista.sort()
                print("LISTA DELLA SPESA: ", lista)
        else:
                print("La lista della spesa è vuota.")
```

La quarta funziona fa a gestire la scrittura di un file di testo nel caso in cui la lista non sia una lista vuota. La riga "***stringa = " ".join(lista)***" va a trasformare la lista della spesa in una stringa senza punti, ***open(nome_file, "w")*** va ad aprire il file in scrittura il quale verrà poi scritto con il metodo ***.write()***

```
def scrivi_file(lista):
        if lista:
                x = input("Inserire il nome del file su cui scrivere la lista, .txt: ")
                stringa = " ".join(lista)
                file_lista = open(x, "w")
                file_lista.write(stringa)
                file_lista.close()
        else:
                print("Non è stato possibile scrivere il file")
```

L' ultima funzione va a caricare la lista direttamente da un file di testo il cui nome viene inserito in input. In questo caso il file verrà aperto in lettura similmente al caso precedente (scrittura) copiando il contenuto in una stringa che verrà poi splittata in una lista mediante il metodo ***.split()***.

```
def carica_lista():
        nome = input("Inserire il nome del file da cui caricare la lista della spesa: ")
        file_lista = open(nome, "r", encoding = "UTF-8")
        stringa = file_lista.read()
        file_lista.close()
        return stringa.split()
```

## *CODICE FINALE*

```
File  Actions  Edit  View  Help
 GNU nano 8.1                                                    bonus.py
#programma gestione lista spesa
def aggiungi_elemento(lista):
        x = input("Aggiungi un elemento: ").strip()
        if x:
                lista.append(x)
                return lista
        else:
                print("Non puo essere aggiunto alla lista")
def rimuovi_elemento(lista):

        x = input("Qaule elemento vuoi rimuovere?").strip()
        if x in lista:
                lista.remove(x)
                print("Rimozione avvenuta")
                return lista
        else:
                print("Non è stato possibile effettuare la rimozione")
def visualizza_lista(lista):
        if lista:
                lista.sort()
                print("LISTA DELLA SPESA: ", lista)
        else:
                print("La lista della spesa è vuota.")
def scrivi_file(lista):
        if lista:
                x = input("Inserire il nome del file su cui scrivere la lista, .txt: ")
                stringa = " ".join(lista)
                file_lista = open(x, "w")
                file_lista.write(stringa)
                file_lista.close()
                print("File scritto correttamente")
        else:
                print("Non è stato possibile scrivere il file")
def carica_lista():
        nome = input("Inserire il nome del file da cui caricare la lista della spesa: ")
        file_lista = open(nome, "r", encoding = "UTF-8")
        stringa = file_lista.read()
        file_lista.close()
        return stringa.split()

lista_spesa = []
print("Il programma permette di gestire una lista della spesa")

print(("Cosa vuoi fare? \n 1: Per aggiungere un elemento \n 2: Per Rimuoverlo\n 3: Per visualizzare la lista\n 4: Per scrivere la lista su un file\n 5: Per caricare la lista da un file."))
while True:
        x = input("Cosa vuoi fare? \b ")
        if x == "1":
                lista_soesa = aggiungi_elemento(lista_spesa)
        elif x == "2":
                lista_spesa = rimuovi_elemento(lista_spesa)
        elif x == "3":
                visualizza_lista(lista_spesa)
        elif x == "4":
                scrivi_file(lista_spesa)
        elif x == "5":
                lista_spesa = carica_lista()
        else:
                print("Chiusura del programma")
```

Nel main verrà aperto un menu che indicherà all' utente cosa sarà possibile fare semplicemente inserendo in input un numero dall' uno al 5 gestito mediante una serie di if-elif.

## *ESECUZIONE*

Segue un esempio di una veloce esecuzione del programma dando in input gli elementi "pasta" e "carne" i quali verrano poi ordinati alfabeticamente, segue la rimozione dell' elmenento "carne" successivamente trascritto sul file esempio.txt.

```
Il programma permette di gestire una lista della spesa
Cosa vuoi fare?
 1: Per aggiungere un elemento
 2: Per Rimuoverlo
 3: Per visualizzare la lista
 4: Per scrivere la lista su un file
 5: Per caricare la lista da un file.
Cosa vuoi fare? 1
Aggiungi un elemento: pasta
Cosa vuoi fare? 1
Aggiungi un elemento: carne
Cosa vuoi fare? 3
LISTA DELLA SPESA:  ['carne', 'pasta']
Cosa vuoi fare? 2
Qaule elemento vuoi rimuovere?carne
Rimozione avvenuta
Cosa vuoi fare? 3
LISTA DELLA SPESA:  ['pasta']
Cosa vuoi fare? 4
Inserire il nome del file su cui scrivere la lista, .txt: esempio.txt
File scritto correttamente
Cosa vuoi fare?
```