

S2/L5

Il progetto di oggi prevede la seguente 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 nel 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

Userà wsl per comodità negli screenshot.

Si considera assodata la sintassi dei comandi da terminale per creare il file .py, la modifica tramite nano e l'esecuzione del programma, quindi si mostrerà solo la lista dei comandi eseguiti:

```
marco@PORTATILE:~/python$ touch oraesatta.py
```

```
nano oraesatta.py
```

```
python3 oraesatta.py
```

ANALISI CODICE

A seguire il codice che viene poi frammentato ed analizzato:

```
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? ")
```

```
    if comando_utente.lower() == "esci":
```

```
        print("Arrivederci!")
```

```
        break
```

```
    else:
```

```
        print(assistente_virtuale(comando_utente))
```

Che fa questo programma?

Guardiamo la funzione principale:

```
while True
    comando_utente = input("Cosa vuoi sapere? ")
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
    else:
        print(assistente_virtuale(comando_utente))
```

Fintanto che non viene inserito dall'utente un input "esci", il programma continua ad essere eseguito.

Invece se viene inserito "esci", il programma stampa "Arrivederci!" e si interrompe.

Con .lower() si evitano problemi di case sensitiveness

Dando una veloce occhiata alla funzione, viene definita una funzione `assistente_virtuale` che prende in input un *comando* sotto forma di variabile. Guardiamo il *comando* che viene preso in ingresso:

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

```
elif comando == "Che ore sono?":
```

La variabile *comando* sembra essere una stringa

Sempre guardando questa riga:

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

Viene fatto un controllo condizionale sull'input e viene confrontato con tre diverse opzioni

La funziona restituisce *risposta* sotto forma di stringa

Quindi riassumendo sembra che si tratti di una funzione che prende in input (probabilmente da utente) un comando e restituisce, usando la libreria `datetime`, l'ora/data/ora+data attuali.

Proviamo a fare un disegno:

```
import datetime
def assistente_virtuale(comando):
    if comando == "Qual è la data di oggi?":
        oggi = datetime.datetime.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

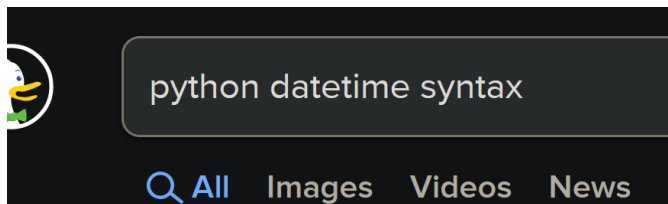
while True:
    comando_utente = input("Cosa vuoi sapere? ")
    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break
    else:
        print(assistente_virtuale(comando_utente))
```

The image shows a Python script for a virtual assistant. The script defines a function `assistente_virtuale` that takes a command as input and returns a response. The function uses `datetime` to get the current date and time. The script then enters a `while True` loop that prompts the user for input. If the user enters "esci", the program prints "Arrivederci!" and breaks the loop. Otherwise, it calls the `assistente_virtuale` function and prints the response. Handwritten annotations in red, purple, and orange highlight various parts of the code: the function definition, the conditional logic, the return statement, and the loop structure.

Errori di sintassi / logici

Prima di correggere i casi non standard, vado a correggere la sintassi e la logica

Ora andrei a vedere la libreria `datetime`. Faccio una rapida ricerca su internet



[Qui link alla documentazione](#)

Inizio a cercare la sintassi relativa alla libreria

```
Other constructors, all class methods:  
  
classmethod date.today()  
    Return the current local date.  
  
This is equivalent to date.fromtimes
```

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

Qui manca un punto, andiamo a correggerlo così:

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

Vediamo altro

```
elif comando == "Che ore sono?":
```

```
    ora_attuale = datetime.datetime.now().time()
```

```
    risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
```

Questo è ok.

Il metodo **.strftime** permette di formattare l'ora

il metodo **.time()** permette di rappresentare l'ora acquisita con datetime senza mettere la data.

```
while True
```

Qui mancano i due punti :

Ecco la forma corretta:

```
while True:
    comando
```

Provando ad eseguire il codice, dopo queste rapide correzioni, il codice sembra funzionare dal punto di vista della sintassi.

La logica sembra ok, il codice si può migliorare in tanti modi, ad esempio un dizionario di variazioni di frasi o di termini che possono essere richiamati per eseguire un comando, ma cerchiamo di mantenere il codice leggero.

L'unica miglioria che farei è, nel caso di input non corrispondente ad un comando (e che manda quindi come output "Non ho capito", si può mettere la lista dei comandi che il programma sa gestire.

Ecco come ho fatto:

```
else:
    risposta = ("Non ho capito la tua domanda. Ecco le frasi a cui so rispondere:\n\n"
               "Come ti chiami?\n"
               "Quale è la data di oggi?\n"
               "Che ore sono?\n")
    return risposta
```

Casistiche non standard

Le casistiche non standard sono i casi in cui il programma ha degli input che non riesce a gestire.

Si può correggere usando solo il lower case, per non avere problemi di case sensitiveness.

Uso il metodo .lower() alla fine di ogni stringa, per confrontarle solo in lower case. Ecco un esempio:

```
if comando == "Qual è la data di oggi?".lower():
    oggi = datetime.date.today()
```

Analogamente lo applico alle altre stringhe di confronto e all'input utente

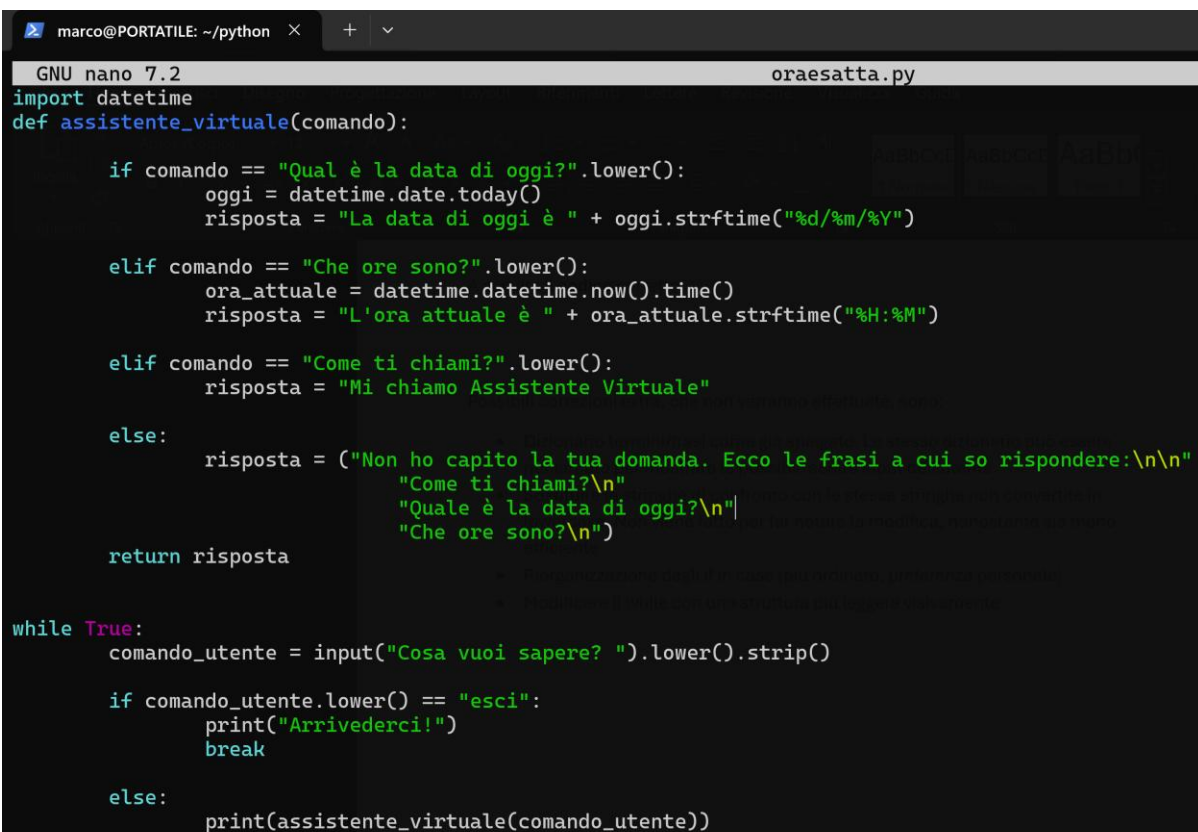
Oltre a questo può avere senso anche usare il metodo .strip() per rimuovere gli spazi extra dall'input

Ecco qui la correzione:

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

Il codice corretto

A seguire il codice corretto:



```
marco@PORTATILE: ~/python x + v
GNU nano 7.2 oraesatta.py
import datetime
def assistente_virtuale(comando):

    if comando == "Qual è la data di oggi?".lower():
        oggi = datetime.date.today()
        risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")

    elif comando == "Che ore sono?".lower():
        ora_attuale = datetime.datetime.now().time()
        risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")

    elif comando == "Come ti chiami?".lower():
        risposta = "Mi chiamo Assistente Virtuale"

    else:
        risposta = ("Non ho capito la tua domanda. Ecco le frasi a cui so rispondere:\n\n"
                    "Come ti chiami?\n"
                    "Quale è la data di oggi?\n"
                    "Che ore sono?\n")

    return risposta

while True:
    comando_utente = input("Cosa vuoi sapere? ").lower().strip()

    if comando_utente.lower() == "esci":
        print("Arrivederci!")
        break

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

Possibili correzioni extra, che non verranno effettuate, sono:

- Dizionario termini/frasi come già spiegato. Lo stesso dizionario può essere richiamato per mostrare le possibili scelte input dell'utente
- Sostituire le stringhe di confronto con le stesse stringhe non convertite in lowercase. Non viene fatto per far notare la modifica, nonostante sia meno efficiente
- Riorganizzazione degli if in case (più ordinato, preferenza personale)
- Modificare il while con una struttura più leggera visivamente