

ESERCIZIO: allenare l'osservazione critica.

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:

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.

Svolgimento:

1. Capire cosa fa il programma senza eseguirlo

Il programma è un **semplice assistente virtuale a riga di comando** che:

- Riceve input dall'utente in un ciclo infinito (while True).
- Risponde a 3 comandi specifici:
 - "Qual è la data di oggi?" → restituisce la data corrente.
 - "Che ore sono?" → restituisce l'orario corrente.
 - "Come ti chiami?" → risponde con un nome predefinito.
- Se l'utente scrive "esci" (in qualunque combinazione di maiuscole/minuscole), il programma termina.
- Per ogni altro comando, risponde "Non ho capito la tua domanda."

2. Individuare casistiche non standard non gestite

Il programma **funziona solo se la domanda corrisponde esattamente** alla stringa prevista (inclusi maiuscole, spazi e punteggiatura). Ecco alcune **casistiche non gestite**:

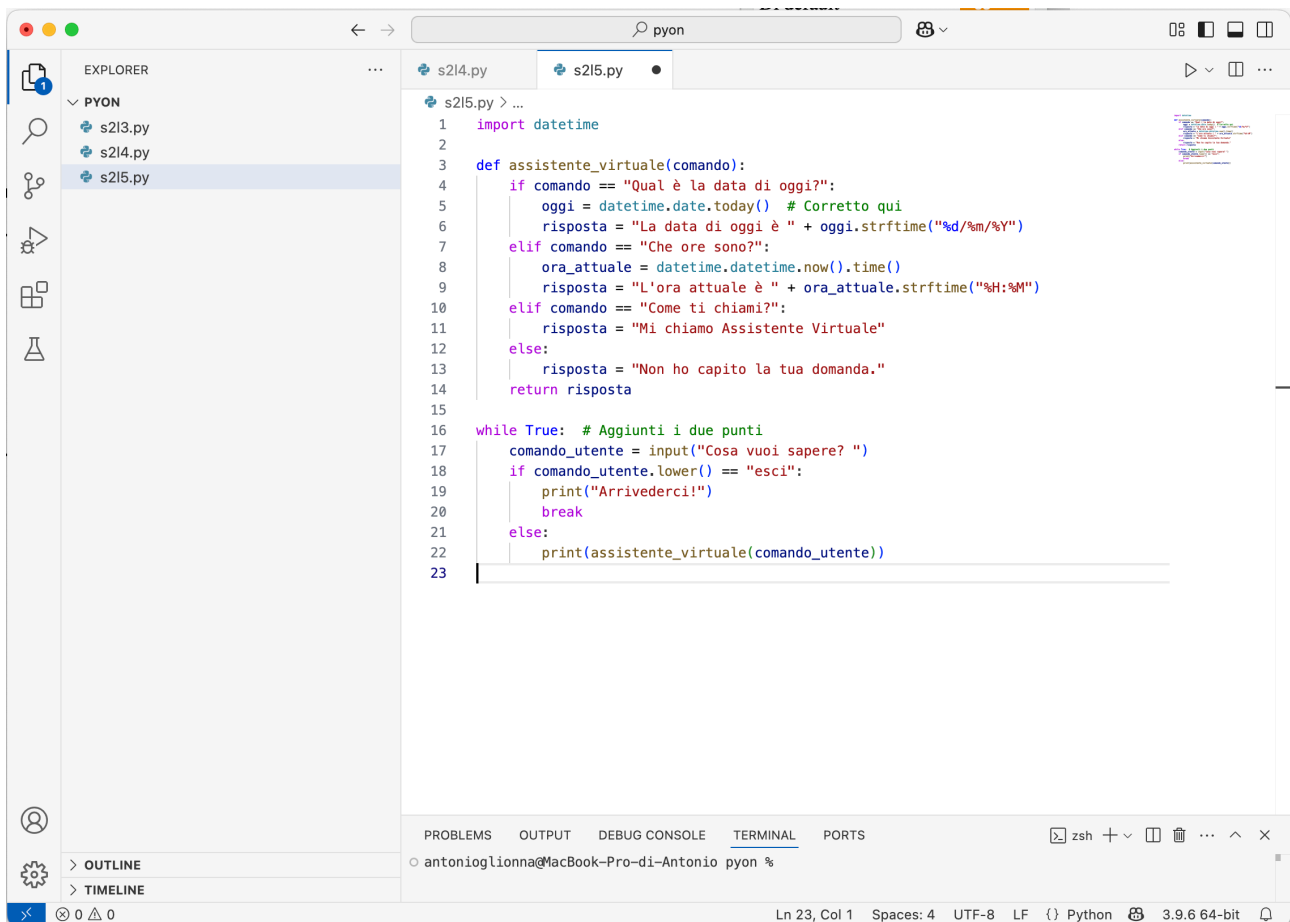
- "Qual è la data di oggi" senza il punto interrogativo.
- "qual è la data di oggi?" tutto minuscolo.

- "Che ore sono" senza il punto.
- "CHE ORE SONO?" tutto maiuscolo.
- Errori di battitura minimi (es. "Qual'è la data di oggi?").
- Inserimento accidentale di spazi extra.

3. Individuare eventuali errori di sintassi / logici

Errori sintattici:

- `datetime.datetoday()` → **Errore**: non esiste questo metodo.
- **Corretto**: `datetime.date.today()`
- `while True` → **Errore**: manca il `:` finale.
- **Corretto**: `while True:`



The screenshot shows a VS Code editor window with a Python file named `s2l5.py` open. The Explorer sidebar on the left shows a project named `PYON` with files `s2l3.py`, `s2l4.py`, and `s2l5.py`. The main editor area displays the following Python code:

```

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

```

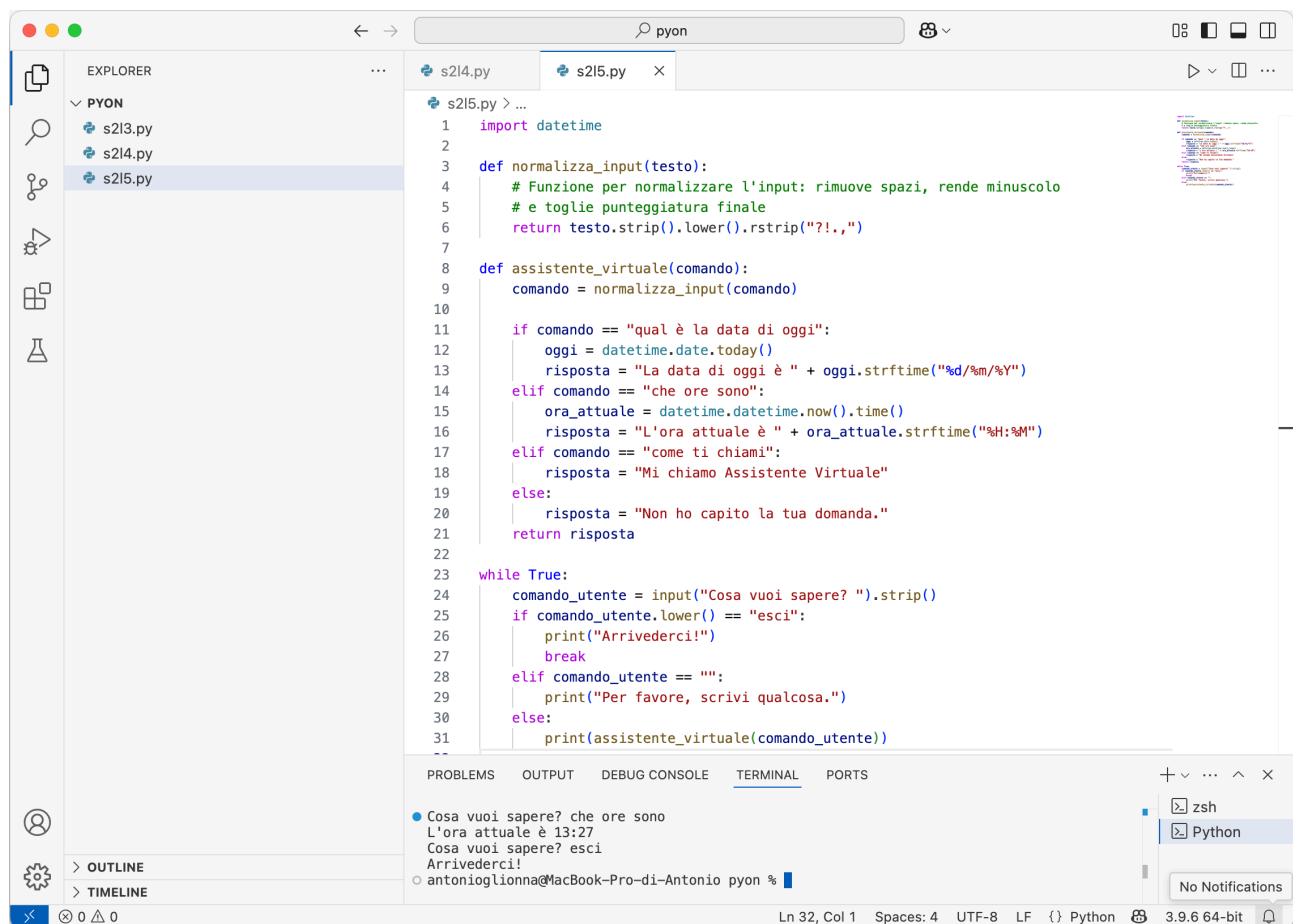
The code contains two syntax errors: `datetime.date.today()` (which should be `datetime.date.today()`) and a missing colon at the end of the `while True:` loop header. The status bar at the bottom indicates the file is `s2l5.py`, line 23, column 1, with 4 spaces, UTF-8 encoding, LF line endings, Python 3.9.6 64-bit.

Errori logici:

- Il confronto if comando == "Qual è la data di oggi?" è **troppo rigido**.
- Non viene gestito alcun controllo sugli **input non standard**.
- Non viene gestito nessun **tipo di eccezione** (es. errore imprevisto).
- Nessuna gestione per input vuoti (invio senza scrivere nulla).

4. Proposta di soluzione per ognuno degli errori

Codice corretto, migliorato e robusto:



```
1 import datetime
2
3 def normalizza_input(testo):
4     # Funzione per normalizzare l'input: rimuove spazi, rende minuscolo
5     # e toglie punteggiatura finale
6     return testo.strip().lower().rstrip("?!.,")
7
8 def assistente_virtuale(comando):
9     comando = normalizza_input(comando)
10
11     if comando == "qual è la data di oggi":
12         oggi = datetime.date.today()
13         risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
14     elif comando == "che ore sono":
15         ora_attuale = datetime.datetime.now().time()
16         risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
17     elif comando == "come ti chiami":
18         risposta = "Mi chiamo Assistente Virtuale"
19     else:
20         risposta = "Non ho capito la tua domanda."
21     return risposta
22
23 while True:
24     comando_utente = input("Cosa vuoi sapere? ").strip()
25     if comando_utente.lower() == "esci":
26         print("Arrivederci!")
27         break
28     elif comando_utente == "":
29         print("Per favore, scrivi qualcosa.")
30     else:
31         print(assistente_virtuale(comando_utente))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

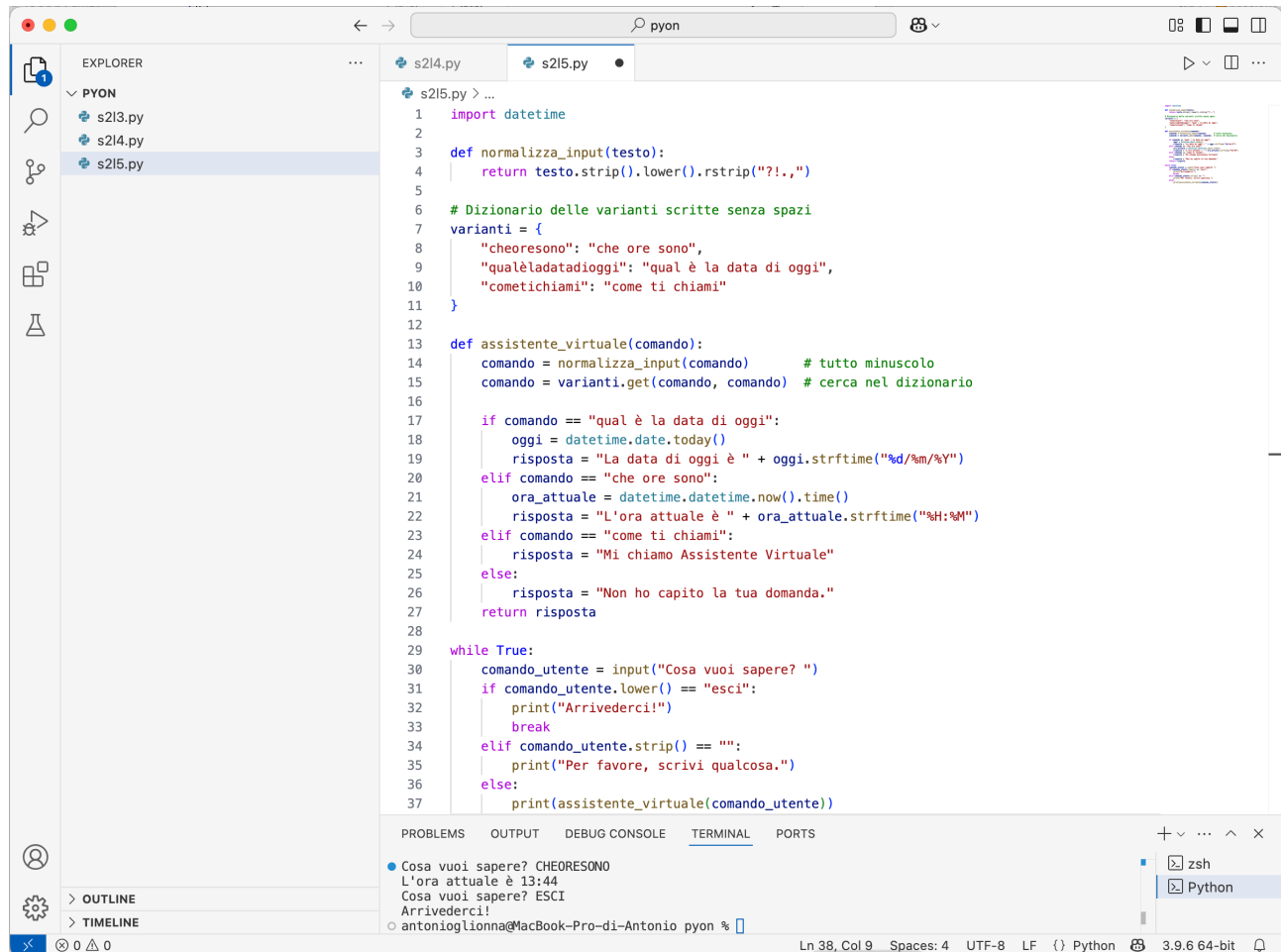
● Cosa vuoi sapere? che ore sono
L'ora attuale è 13:27
Cosa vuoi sapere? esci
Arrivederci!
○ antonioglionna@MacBook-Pro-di-Antonio pyon %

zsh
Python
No Notifications

Ln 32, Col 1 Spaces: 4 UTF-8 LF {} Python 3.9.6 64-bit

Il problema del codice qui sopra anche se corretto e migliorato è che l'assistente virtuale riconosce solo comandi scritti in modo perfetto, con spazi esatti. Questo lo rende incapace di comprendere input come "cheoresono" o "qualèladatadioggi".

Per risolvere, è stato aggiunto un dizionario con varianti comuni scritte senza spazi, che traduce questi comandi in forme comprensibili. In questo modo l'assistente diventa più flessibile e "intelligente", migliorando l'interazione con l'utente.



```
1 import datetime
2
3 def normalizza_input(testo):
4     return testo.strip().lower().rstrip("?!.,")
5
6 # Dizionario delle varianti scritte senza spazi
7 varianti = {
8     "cheoresono": "che ore sono",
9     "qualèladatadioggi": "qual è la data di oggi",
10    "cometichiami": "come ti chiami"
11 }
12
13 def assistente_virtuale(comando):
14     comando = normalizza_input(comando) # tutto minuscolo
15     comando = varianti.get(comando, comando) # cerca nel dizionario
16
17     if comando == "qual è la data di oggi":
18         oggi = datetime.date.today()
19         risposta = "La data di oggi è " + oggi.strftime("%d/%m/%Y")
20     elif comando == "che ore sono":
21         ora_attuale = datetime.datetime.now().time()
22         risposta = "L'ora attuale è " + ora_attuale.strftime("%H:%M")
23     elif comando == "come ti chiami":
24         risposta = "Mi chiamo Assistente Virtuale"
25     else:
26         risposta = "Non ho capito la tua domanda."
27     return risposta
28
29 while True:
30     comando_utente = input("Cosa vuoi sapere? ")
31     if comando_utente.lower() == "esci":
32         print("Arrivederci!")
33         break
34     elif comando_utente.strip() == "":
35         print("Per favore, scrivi qualcosa.")
36     else:
37         print(assistente_virtuale(comando_utente))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● Cosa vuoi sapere? CHEORESONO
L'ora attuale è 13:44
Cosa vuoi sapere? ESCI
Arrivederci!
antonio@MacBook-Pro-di-Antonio pyon %
```

Conclusione

Questo esercizio ha mostrato come un semplice assistente virtuale possa fallire nel riconoscere comandi se non si considera la varietà di input possibili da parte dell'utente. Abbiamo individuato problemi legati a **maiuscole**, **assenza di spazi** e **punteggiatura**, e li abbiamo risolti con una funzione di **normalizzazione dell'input** e un dizionario di **varianti comuni**.

Il risultato è un assistente più flessibile, capace di comprendere input scritti in modi diversi, proprio come farebbe un vero sistema intelligente. Pensare "fuori dagli schemi", proprio come un hacker, significa anche prevedere questi comportamenti e costruire soluzioni resilienti.