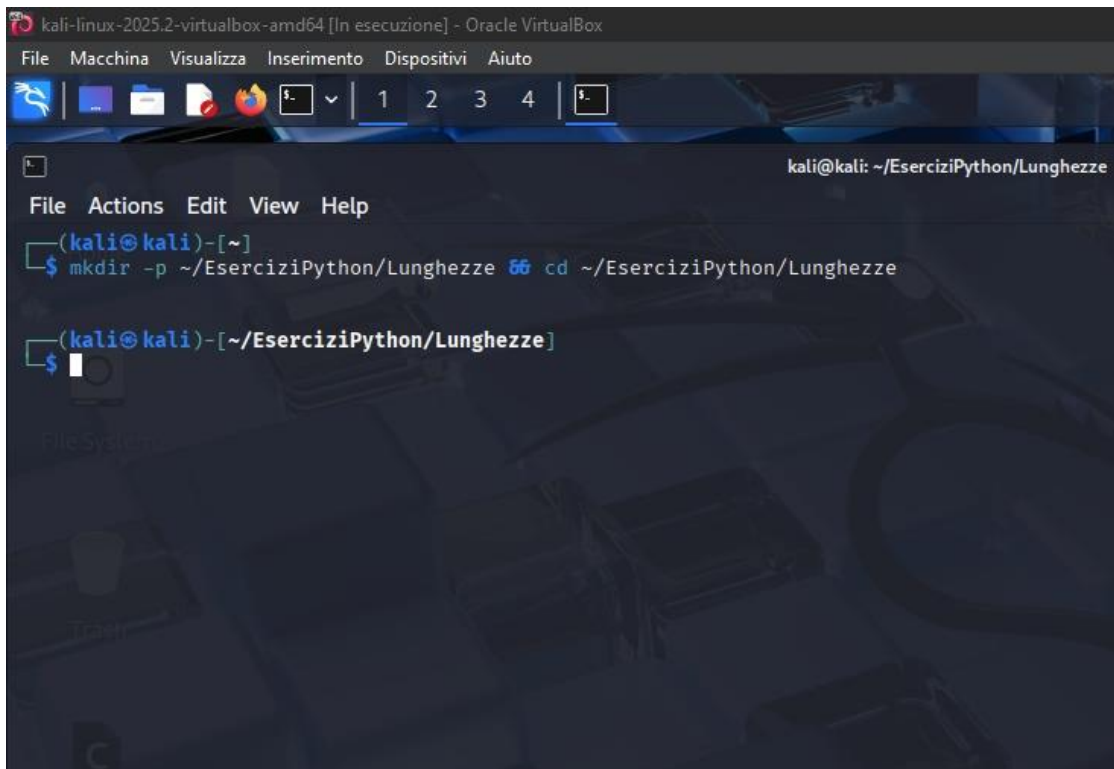


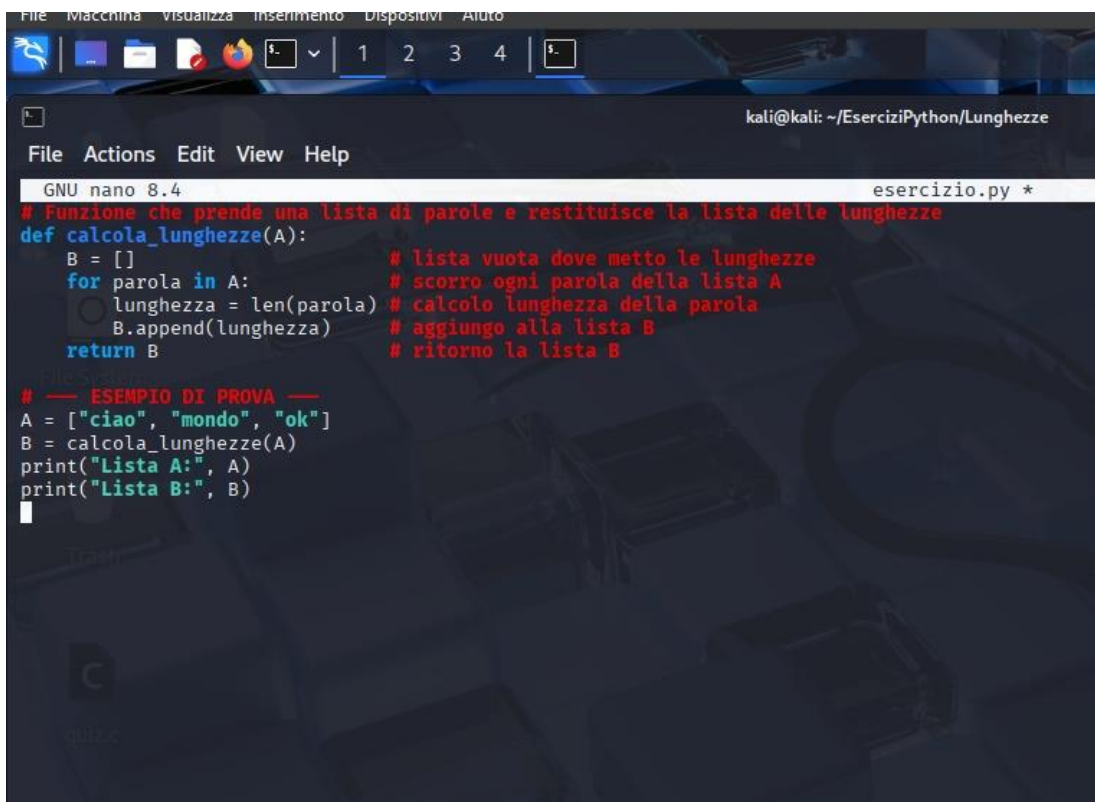
W7D1- FRANCESCO MONTALTO

1) Ho creato una cartella di lavoro, per mantenere l'ordine, tramite i comandi *mkdir -p*, e *cd*.



```
kali@kali: ~/EserciziPython/Lunghezze
File Macchina Visualizza Inserimento Dispositivi Aiuto
$ mkdir -p ~/EserciziPython/Lunghezze && cd ~/EserciziPython/Lunghezze
(kali@kali)~[~/EserciziPython/Lunghezze]
$
```

2) Ho creato il file con nano, tramite il comando “nano esercizio.py”



```
File Macchina Visualizza Inserimento Dispositivi Aiuto
GNU nano 8.4 esercizio.py *
# Funzione che prende una lista di parole e restituisce la lista delle lunghezze
def calcola_lunghezze(A):
    B = [] # lista vuota dove metto le lunghezze
    for parola in A: # scorro ogni parola della lista A
        lunghezza = len(parola) # calcolo lunghezza della parola
        B.append(lunghezza) # aggiungo alla lista B
    return B # ritorno la lista B

# --- ESEMPIO DI PROVA ---
A = ["ciao", "mondo", "ok"]
B = calcola_lunghezze(A)
print("Lista A:", A)
print("Lista B:", B)
```

3) Da qui ho strutturato il testo tramite un commento iniziale (preceduto da "#") per avere chiara l'idea di cosa il successivo codice andrà ad eseguire.

-Le funzioni vere e proprie le ho elaborate tramite il comando "def". Nel mio caso, "calcola_lunghezze" è il nome che ho dato alla funzione. E' seguito dal parametro d'ingresso "A", scritto tra parentesi, che riceve il valore che passeremo quando chiameremo la funzione. Rappresenta la lista di parole di cui vogliamo calcolare la lunghezza.

- "B" è, invece, la lista di output della funzione: contiene le lunghezze delle parole presenti nella lista di input. Le parentesi quadre successive indicano la sua funzione di "elenco dei valori". All'inizio è una lista vuota, ma verrà riempita nell'output.

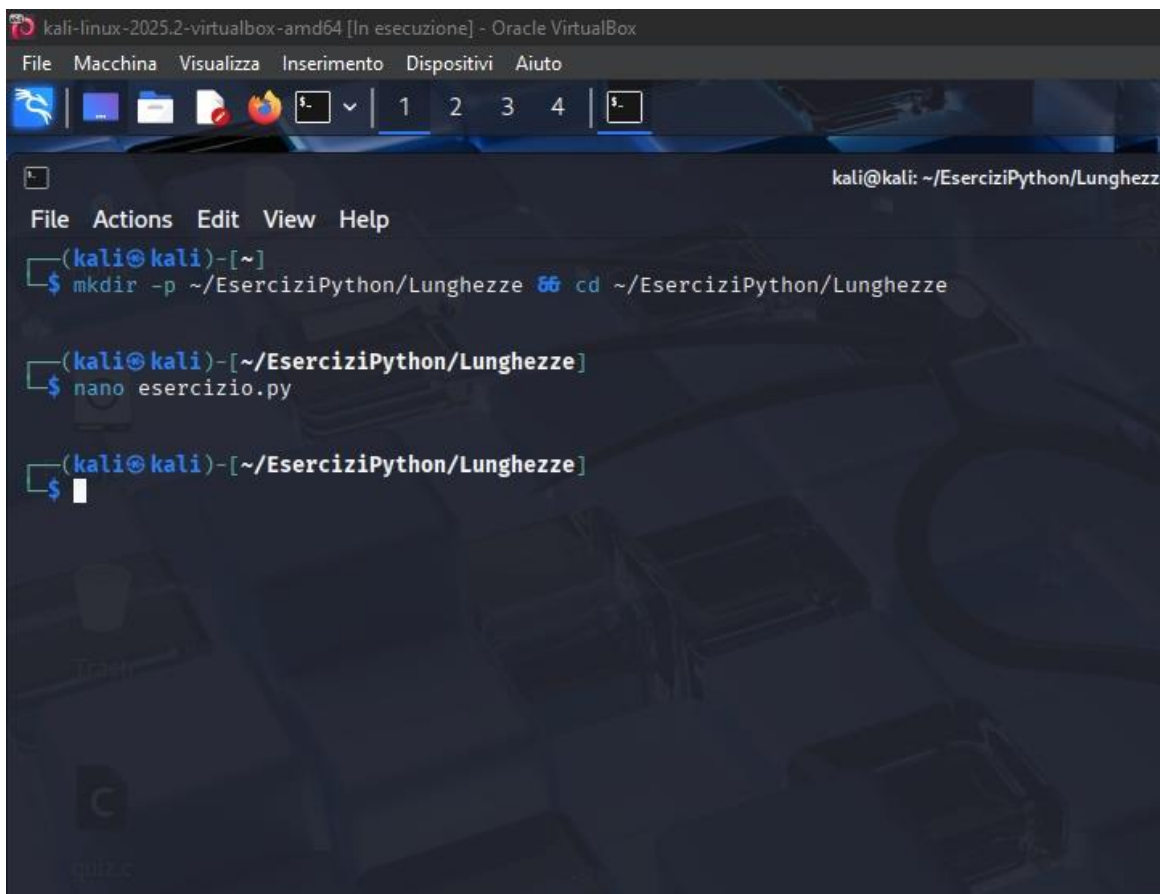
-Ho successivamente utilizzato in comandi "for" (per la ripetizione ciclica delle istruzioni) e "in" (che funge da collante per la lista in cui si deve lavorare).

-All'interno del ciclo, "len(parola)" calcola il numero dei caratteri della parola. "Len" è una funzione che calcola la lunghezza (n° elementi) di un oggetto. "B.append" aggiunge quel numero in fondo alla lista B.

-Infine, tramite il comando "return", ho indicato la risposta finale prodotta dalla funzione. E' seguito da "B", in quanto voglio che il risultato finale sia propriamente quella lista.

Ho testato la funzione sotto la dicitura "esempio di prova".

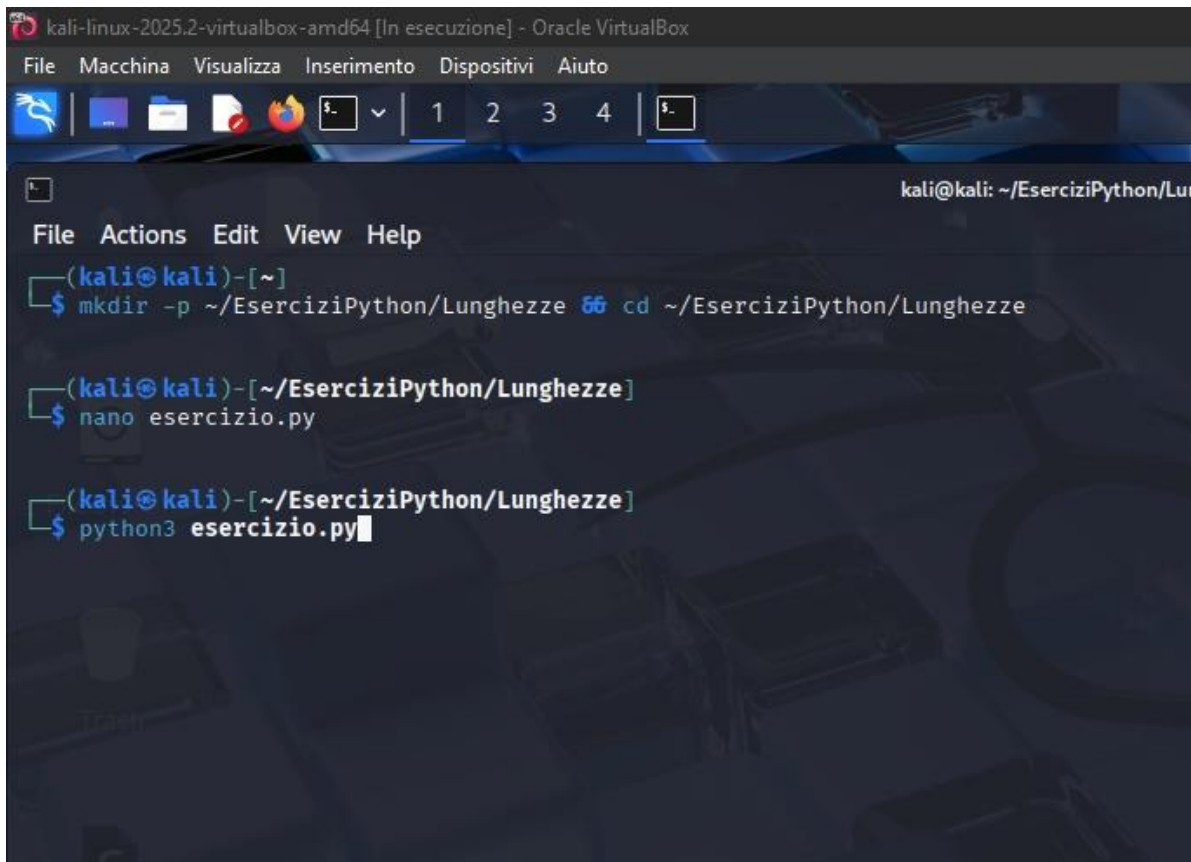
4) Ho successivamente salvato e sono uscito da Nano.



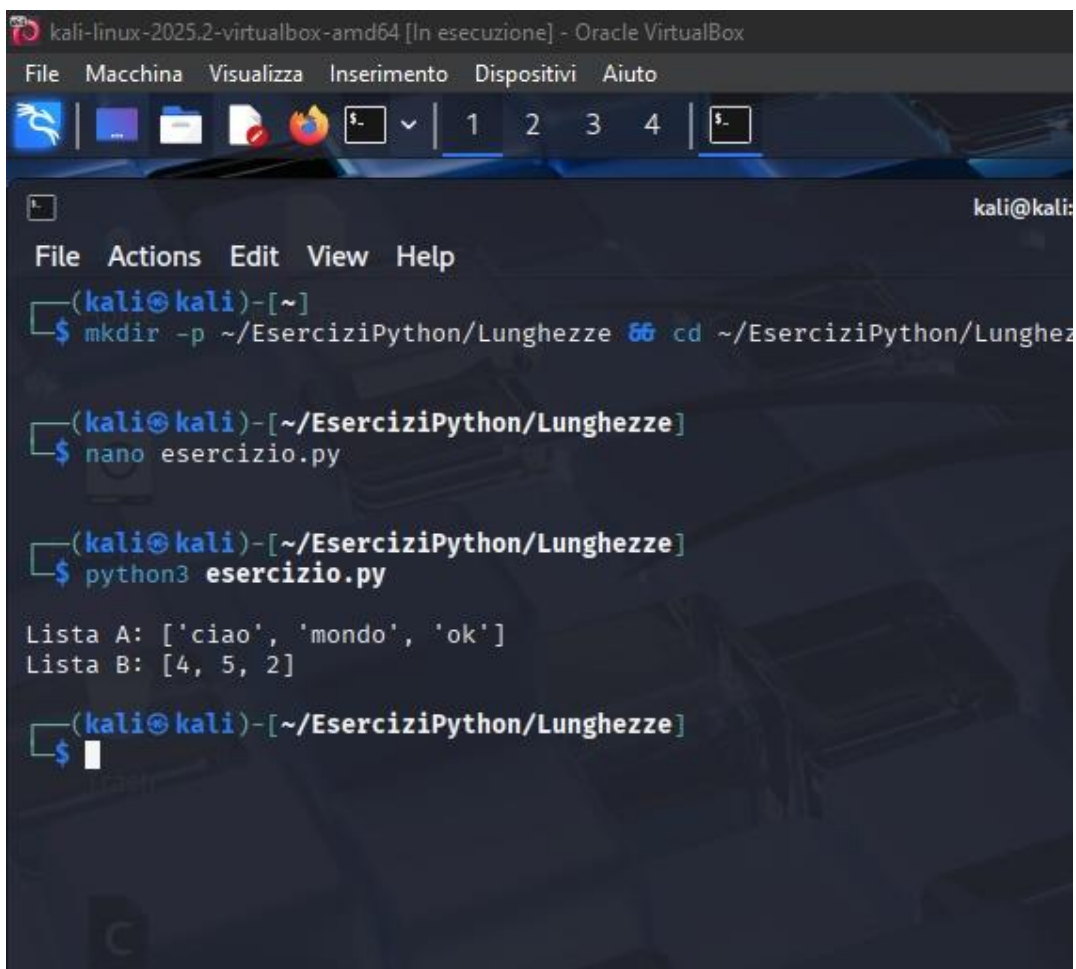
```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto

kali@kali: ~/EserciziPython/Lunghezze
File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Lunghezze && cd ~/EserciziPython/Lunghezze
(kali@kali)-[~/EserciziPython/Lunghezze]
$ nano esercizio.py
(kali@kali)-[~/EserciziPython/Lunghezze]
$
```

5) Ho eseguito il programma tramite il comando “python3 esercizio.py”. Il comando ha prodotto i risultati della lista B.



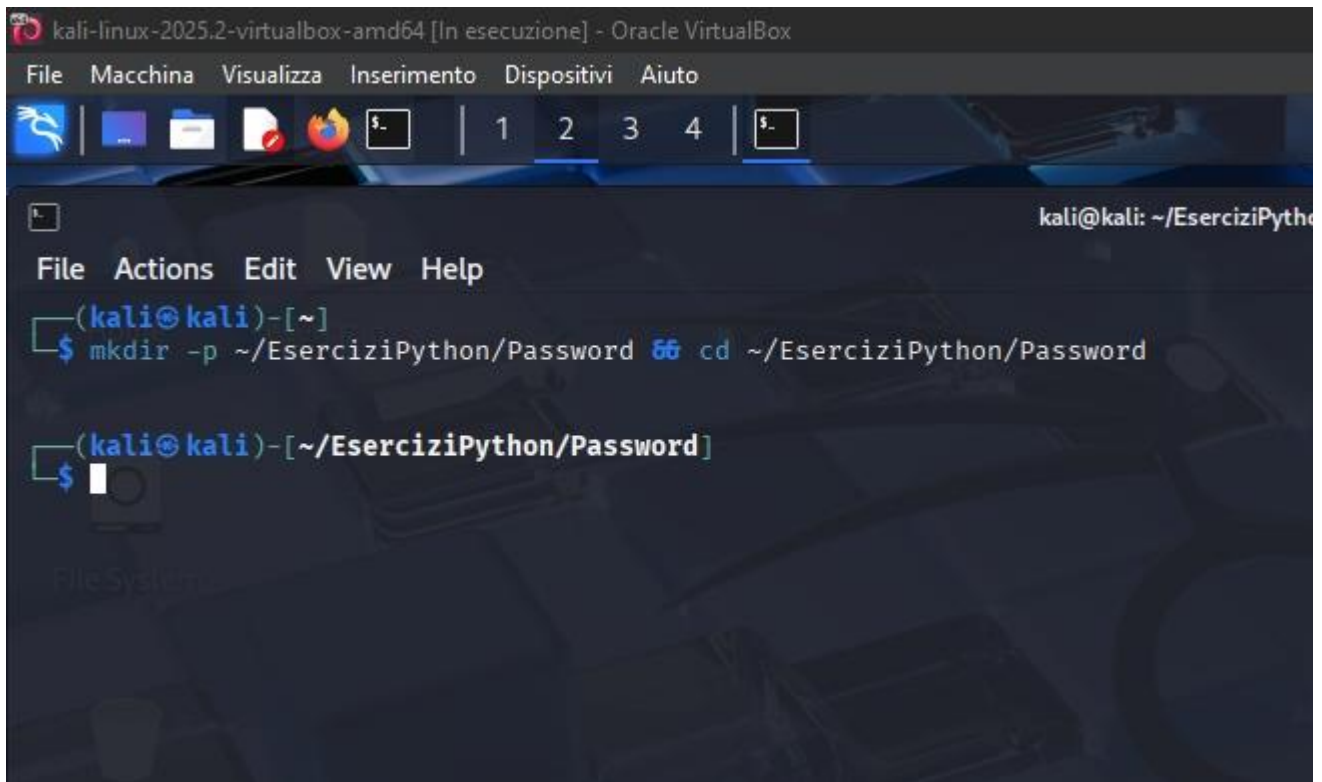
```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
[Icons] | [Terminal] | [Folder] | [File] | [Firefox] | [Terminal] | [Terminal] | [Terminal] | [Terminal] | [Terminal]
kali@kali: ~/EserciziPython/Lunghezze
File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Lunghezze && cd ~/EserciziPython/Lunghezze
(kali@kali)-[~/EserciziPython/Lunghezze]
$ nano esercizio.py
(kali@kali)-[~/EserciziPython/Lunghezze]
$ python3 esercizio.py
```



```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
[Icons] | [Terminal] | [Folder] | [File] | [Firefox] | [Terminal] | [Terminal] | [Terminal] | [Terminal] | [Terminal]
kali@kali: ~/EserciziPython/Lunghezze
File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Lunghezze && cd ~/EserciziPython/Lunghezze
(kali@kali)-[~/EserciziPython/Lunghezze]
$ nano esercizio.py
(kali@kali)-[~/EserciziPython/Lunghezze]
$ python3 esercizio.py
Lista A: ['ciao', 'mondo', 'ok']
Lista B: [4, 5, 2]
(kali@kali)-[~/EserciziPython/Lunghezze]
$
```

ESERCIZIO FACOLTATIVO

1) Seguendo i passi dell'esercizio precedente, ho creato una cartella di lavoro e ci sono entrato dentro, tramite i comandi `mkdir -p` e `cd`.



```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
[Icons] | 1 | 2 | 3 | 4 | [Terminal Icon]

kali@kali: ~/EserciziPython
File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Password && cd ~/EserciziPython/Password

(kali@kali)-[~/EserciziPython/Password]
$
```

2) Ho anche qui aperto l'editor di testo Nano tramite il comando `"nano password_gen.py"`.

Poi ho strutturato il codice della funzione come segue:

-Dopo il commento iniziale preceduto da `"#"` ho scritto l' `"import"` delle librerie; **"random"**, che fornisce valori casuali (in questo caso la casualità della generazione sarà rivolta alle password) e **"string"**, che è una libreria predefinita di python contenente ascii letters, numeri, tutti i caratteri stampabili e tutti i segni di punteggiatura.

-Il comando `"def"` indica che sto definendo una funzione, anche qui. E' seguito da un parametro opzionale che ho denominato `"tipo semplice"`. Il commento multilineare sottostante serve per spiegare quali valori può avere `"tipo"` e ciò che la funzione andrà poi a restituire all'effettivo, ossia la stringa con la password. `"Semplice"` e `"complessa"` sono state strutturate secondo la traccia dell'esercizio.

-Seguitamente al commento, `"if"` (che significa `"se"`) è seguito da `"=="`, che serve a verificare l'uguaglianza (da non confondere con `"="`, che è un valore di assegnazione) e seguito da `"semplice"`. Volendo una password semplice, io utente voglio una lunghezza di 8 caratteri composta di stringhe e numeri.

-Sotto, `"elif"` (`"altrimenti"`), serve a controllare un'altra condizione, a patto che quella posta da `"if"` sia falsa. Qui ho scritto `"complessa"`, perché ne ho imposto una lunghezza di 20 tra tutti i caratteri possibili.

-In ultimo, `"else"` (`"altro"`) solleva un errore (`"raise ValueError"`, in cui `"raise"` è un comando di sollevamento di eccezione, e `"ValueError"` è un predefinito di Python per indicare valori non validi), nel caso in cui ambedue le precedenti fossero errate, seguita quindi da `"tipo non valido: usa semplice o complessa"`.

- La linea che segue è quella che genera la password vera e propria:
`random.choice(caratteri)`, che sceglie un carattere casuale tra tutti quelli disponibili.
- `for _ in range(lunghezza)`, che è un ciclo `for` in forma compatta. `_` serve per ripetere il ciclo e `range(lunghezza)` crea una sequenza che parte da 0 a n-1. Il ciclo si ripete tante volte quanto la lunghezza desiderata della password (8 o 20).
- Successivamente `"".join(...)`, è composto da due paia di virgolette (`" "`) che indica che vogliamo unire i caratteri senza nulla tra di loro (nessuno spazio) e `"".join(...)` che prende tutti i caratteri prodotti dal ciclo e li concatena in un'unica stringa.
- `Return Password` serve per ridarci indietro la stringa che abbiamo appena generato, cosicché chi genera la funzione può tranquillamente stamparla o salvarla.
- Successivamente all'ultimo commento, ho scritto una variabile speciale chiamata `"__name__"`, che se eseguito direttamente assume il valore di `"__main__"`, altrimenti vale il nome del file se importato in un altro script. In questo caso è eseguibile solo se il file è il programma principale.
- Ho salvato l'input che ritenevo più opportuno nella variabile `"tipo scelto"`, così che l'utente possa scrivere `"semplice"` o `"complessa"`.
- Ho poi chiamato la funzione precedentemente definita, passando il tipo scelto dall'utente. Questa funzione genera la password e la restituisce con `"return"`. Ho salvato questo valore nella variabile `"pwd"` (abbreviazione di `"password"`).
- Il finale comando `"print"` stampa a schermo la password generata.

```

File Actions Edit View Help
GNU nano 8.4 password_gen.py *
# Funzione per generare password casuali
import random
import string

def genera_password(tipo="semplice"):
    """
    tipo: "semplice" -> password 8 caratteri alfanumerici
         "complessa" -> password 20 caratteri ASCII stampabili
    Ritorna: stringa con la password generata
    """
    if tipo == "semplice":
        caratteri = string.ascii_letters + string.digits # lettere + numeri
        lunghezza = 8
    elif tipo == "complessa":
        caratteri = string.printable.strip() # tutti i caratteri stampabili, senza spazi finali
        lunghezza = 20
    else:
        raise ValueError("Tipo non valido: usa 'semplice' o 'complessa'")

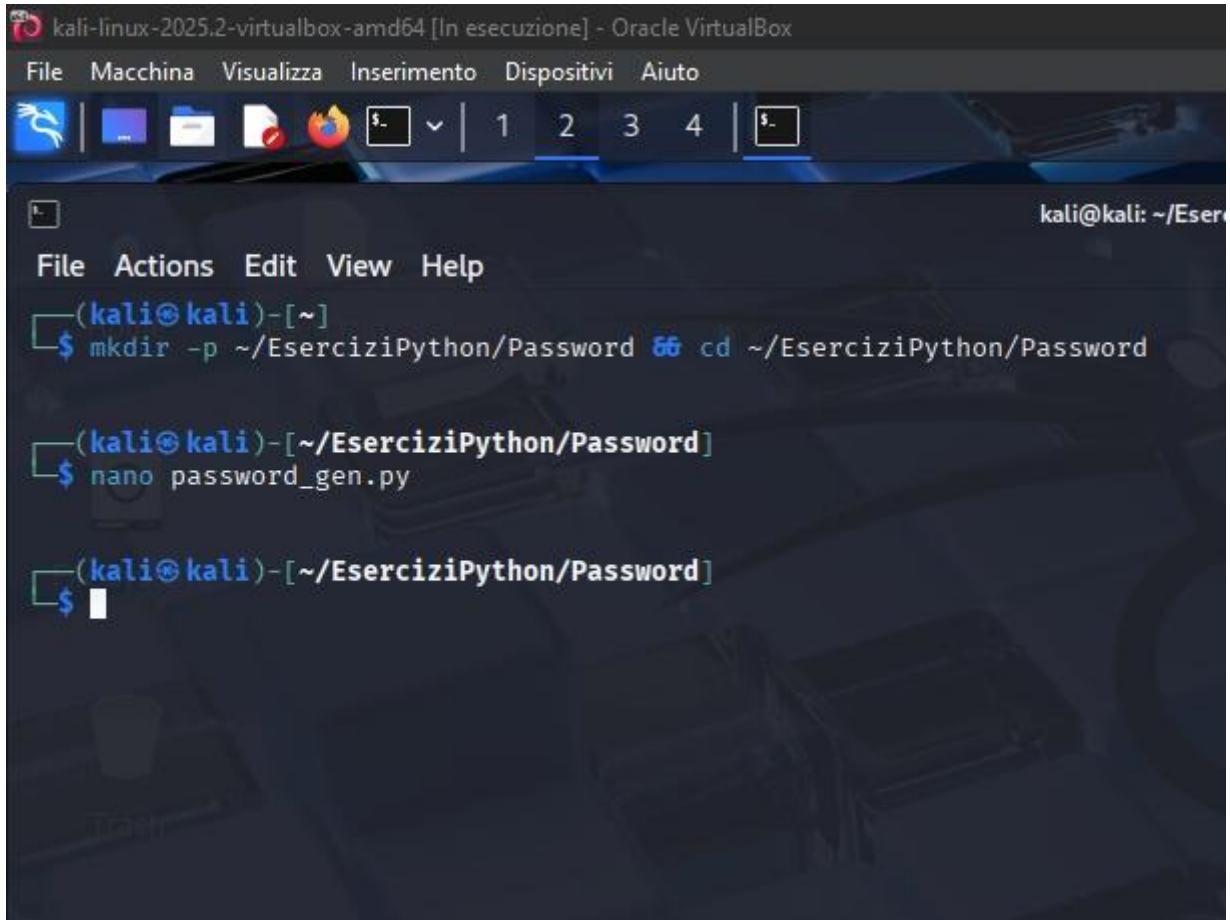
    password = "".join(random.choice(caratteri) for _ in range(lunghezza))
    return password

# Esempio di utilizzo
if __name__ == "__main__":
    tipo_scelto = input("Vuoi una password semplice o complessa? ")
    pwd = genera_password(tipo_scelto)
    print("La tua password:", pwd)

^G Help      ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line  M-E

```

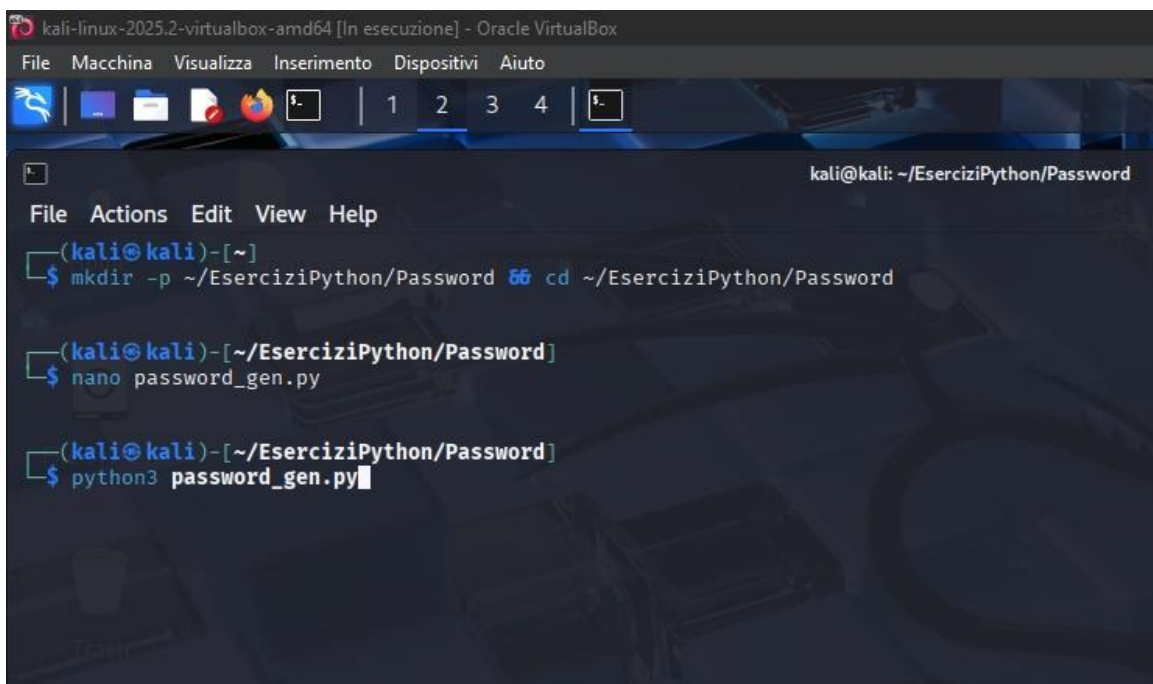
4) Ho salvato e chiuso.



```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File Macchina Visualizza Inserimento Dispositivi Aiuto

(kali@kali)~$ mkdir -p ~/EserciziPython/Password && cd ~/EserciziPython/Password
(kali@kali)~/EserciziPython/Password$ nano password_gen.py
(kali@kali)~/EserciziPython/Password$
```

5) Ho successivamente eseguito il programma con “python3 password_gen.py”, e questo testato quanto seguiva.



```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File Macchina Visualizza Inserimento Dispositivi Aiuto

(kali@kali)~$ mkdir -p ~/EserciziPython/Password && cd ~/EserciziPython/Password
(kali@kali)~/EserciziPython/Password$ nano password_gen.py
(kali@kali)~/EserciziPython/Password$ python3 password_gen.py
```

d

```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
[Icone] | [1] [2] [3] [4] [5]

kali@kali: ~/EserciziPython/Password

File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Password && cd ~/EserciziPython/Password

(kali@kali)-[~/EserciziPython/Password]
$ nano password_gen.py

(kali@kali)-[~/EserciziPython/Password]
$ python3 password_gen.py

Traceback (most recent call last):
  File "/home/kali/EserciziPython/Password/password_gen.py", line 24, in <module>
    if name == "_main_":
       ^^^^^
NameError: name '_name_' is not defined. Did you mean: '__name__'?

(kali@kali)-[~/EserciziPython/Password]
$ nano password_gen.py

(kali@kali)-[~/EserciziPython/Password]
$ python3 password_gen.py

Vuoi una password semplice o complessa? █
```

```
kali-linux-2025.2-virtualbox-amd64 [In esecuzione] - Oracle VirtualBox
File  Macchina  Visualizza  Inserimento  Dispositivi  Aiuto
[Icone] | [1] [2] [3] [4] [5]

kali@kali: ~/EserciziPython/Password

File  Actions  Edit  View  Help
(kali@kali)-[~]
$ mkdir -p ~/EserciziPython/Password && cd ~/EserciziPython/Password

(kali@kali)-[~/EserciziPython/Password]
$ nano password_gen.py

(kali@kali)-[~/EserciziPython/Password]
$ python3 password_gen.py

Traceback (most recent call last):
  File "/home/kali/EserciziPython/Password/password_gen.py", line 24, in <module>
    if name == "_main_":
       ^^^^^
NameError: name '_name_' is not defined. Did you mean: '__name__'?

(kali@kali)-[~/EserciziPython/Password]
$ nano password_gen.py

(kali@kali)-[~/EserciziPython/Password]
$ python3 password_gen.py

Vuoi una password semplice o complessa? complessa
La tua password: CBSLUm@]: "DUC@'c!?Q$

(kali@kali)-[~/EserciziPython/Password]
$ █
```