Programmazione per Hacker – Python pt. 2

W7D1

Ho incluso tutta la spiegazione dei passaggi nel codice come suggerito in call (In Verde)

home > kali > progetti > W7D1.py > ...

```python
# Definiamo la funzione lunghezza_parole che accetta una lista A di parole come input
def lunghezza_parole(A):
    # Creo una lista vuota B dove metterò le lunghezze delle parole
    B = []

    # Faccio un ciclo per guardare ogni parola nella lista A
    for parola in A:
        # Calcolo la lunghezza della parola corrente usando len()
        lunghezza = len(parola)
        # Aggiungo la lunghezza trovata alla lista B
        B.append(lunghezza)

    # Ritorno la lista B con tutte le lunghezze
    return B

# Creo una lista di esempio A con alcune parole per provare la funzione
A = ["ciao", "python", "hacker", "code", "Epicode"]
# Chiamo la funzione lunghezza_parole con la lista A e salvo il risultato
risultato = lunghezza_parole(A)
# Stampo il risultato, che dovrebbe essere una lista con le lunghezze
print(risultato)
```

Test e descrizione

Ho svolto il test sul terminale per verificare che soddisfasse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio:

EPICODE

Esercizio
Python per Hacker Pt. 2

Traccia:

Scrivi una funzione che data in ingresso una lista A contenente n parole, restituisca in output una lista B di interi che rappresentano la lunghezza delle parole contenute in A.

Rappresento di seguito da terminale tutte le lunghezze di ogni parola:

4 = ciao _ 6=python _ 6=hacker _ 4=code _ 7=Epicode

```
┌──(kali㉿kali)-[~/progetti]
└─$ python3 W7D1.py
[4, 6, 6, 4, 7]
```

FACOLTATIVO

W7D1 Facoltativo

Ho incluso tutta la spiegazione dei passaggi nel codice come suggerito in call (In Verde)

home > kali > progetti > W7D1_Facoltativo.py > ...

```python
# Importiamo il modulo random per generare caratteri casuali
import random
# Importiamo string per accedere a lettere, numeri e caratteri ASCII
import string

# Definiamo la funzione genera_password che accetta un parametro tipo ("semplice" o "complicata")
def genera_password(tipo="semplice"):
    # Se il tipo è "semplice", io genero una password alfanumerica di 8 caratteri
    if tipo == "semplice":
        # Mi preparo usando lettere maiuscole, minuscole e numeri come caratteri
        caratteri = string.ascii_letters + string.digits
        # Creo la password di 8 caratteri scegliendo casualmente tra i caratteri
        password = ''.join(random.choice(caratteri) for _ in range(8))
    # Se il tipo è "complicata", io genero una password ASCII di 20 caratteri
    else:
        # Aggiungo anche la punteggiatura per rendere la password più complessa
        caratteri = string.ascii_letters + string.digits + string.punctuation
        # Creo la password di 20 caratteri scegliendo casualmente tra tutti i caratteri
        password = ''.join(random.choice(caratteri) for _ in range(20))
    # Ritorno la password che ho generato
    return password

# Faccio un test con una password semplice
print("Password semplice:", genera_password("semplice"))
# Faccio un test con una password complicata
print("Password complicata:", genera_password("complicata"))
```

Test e descrizione

Ho svolto il test sul terminale per verificare che soddisfasse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio

EPICODE

Esercizio
Python per Hacker Pt. 2

Facoltativo:

Scrivi una funzione generatrice di password.

La funzione deve generare una stringa alfanumerica di 8 caratteri qualora l'utente voglia una password semplice, o di 20 caratteri ascii qualora desideri una password più complicata.

Rappresento di seguito tutte le due tipologie di password su terminale:

```
┌──(kali㉿kali)-[~/progetti]
└─$ python3 W7D1_Facoltativo.py
Password semplice: YTiI1Hxz
Password complicata: ;ME5|J`e"'7JvMm[kmg`
```

SUPER FACOLTATIVO

W7D1

Creare un mini-server che accetta connessioni sulla porta 8888 e riceve delle parole.
Il mini-sever risponde con la lunghezza delle stesse

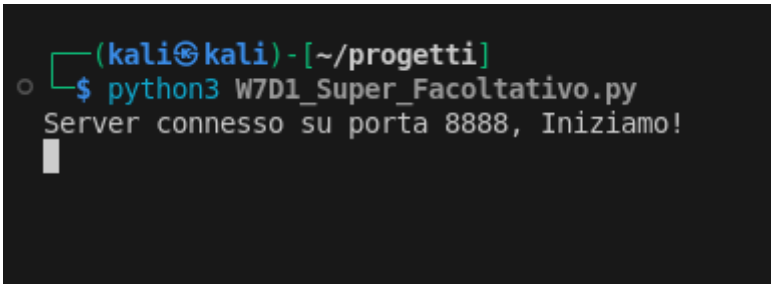
home > kali > progetti > W7D1_Super_Facoltativo.py > start_server

```python
import socket

def start_server():
    # Creo il mio server con socket, lo faccio girare sulla porta 8888!
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Lo lego a 127.0.0.1, così resto sul mio PC, ma posso cambiarlo in '0.0.0.0' se voglio testare in rete.
    server.bind(('127.0.0.1', 8888))
    # Dico al server di accettare una connessione alla volta
    server.listen(1)
    # Metto un messaggio per sapere che è partito, così sono sicuro
    print("Server connesso su porta 8888, Iniziamo!")

    while True:
        # Aspetto che qualcuno si connetta, accetto la chiamata
        conn, addr = server.accept()
        # Mando un menu al client per scegliere cosa fare
        menu = "Scegli un'opzione:\n1. Totale (lunghezza totale)\n2. Parole (lunghezze delle parole)\n3. Caratteri unici (conteggio caratteri distinti)\nInvia il numero: "
        conn.sendall(menu.encode())

        # Ricevo la scelta del client
        scelta = conn.recv(1024).decode().strip()
        # Prendo i dati (parole) dal client dopo la scelta
        data = conn.recv(1024).decode().strip()
        # Se non arriva niente, salto e chiudo
        if not data:
            conn.close()
            continue

        # Decido cosa fare in base alla scelta
        if scelta == "1":
            # Calcolo la lunghezza totale, tolgo spazi extra se ci sono
            lunghezza_totale = len(data.strip())
            # Lo trasformo in stringa e lo mando
            risposta = str(lunghezza_totale)
        elif scelta == "2":
            # Divido la stringa in parole
            parole = data.split()
            # Faccio la lunghezza di ogni parola
            lunghezze = [len(parola) for parola in parole]
            # Unisco tutto con spazi, pronto per inviarlo
            risposta = " ".join(map(str, lunghezze))
        elif scelta == "3":
            # Conto i caratteri unici nella stringa, trasformo in set
            caratteri_unici = len(set(data.replace(" ", "")))
            # Invio il conteggio
            risposta = str(caratteri_unici)
        else:
            # Se la scelta è sbagliata, uso la lunghezza totale come default
            lunghezza_totale = len(data.strip())
            risposta = str(lunghezza_totale)

        # Mando la risposta con sendall, così sono sicuro che arriva
        conn.sendall(risposta.encode())
        # Chiudo la connessione, finito con questo client
        conn.close()

# Avvio il test del mio server mixato!
start_server()
```

Test e descrizione

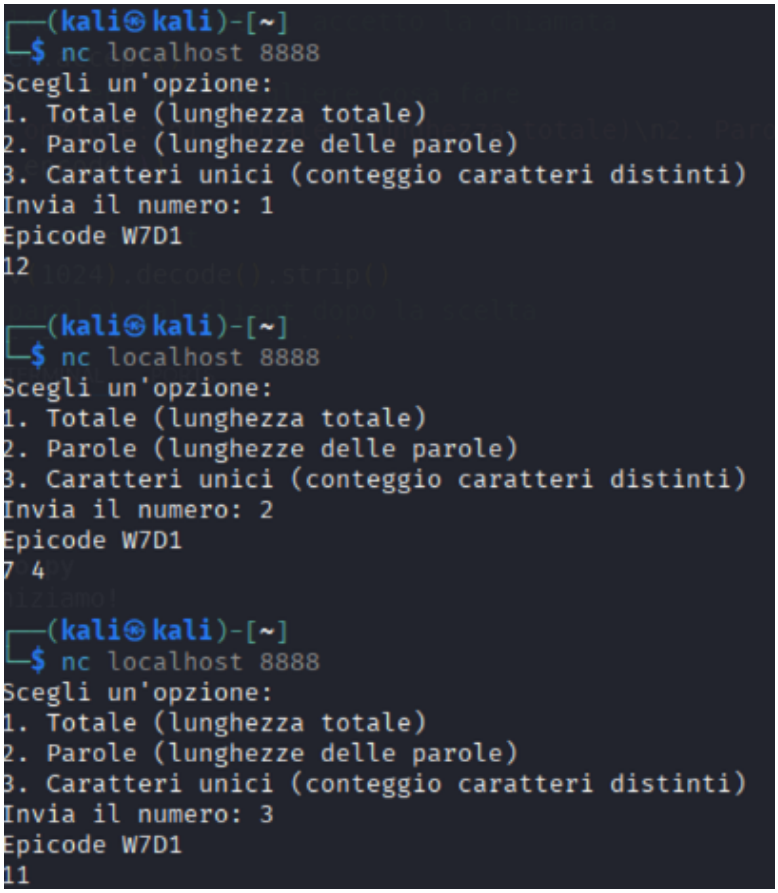
Ho svolto il test sul terminale per verificare che soddisfasse tutti i requisiti richiesti dall'esercizio, allegando il test di seguito dopo la traccia dell'esercizio

- Ho aperto il primo terminale.
- Ho effettuato la connessione e lasciato in esecuzione il server su porta 8888



- Ho aperto il secodo terminale lasciando aperto anche il primo

- Ho digitato : nc localhost 8888 è avviato il programma, di seguito risultato test



Descrizione delle funzionalità del mio mix

Mentro scrivevo il mio codice nel test ho notato che era semplice, mi calcolava la lunghezza totale dei byte.

Vedendo poi il codice del Prof era più complesso cosi ho deciso di mixarlo in modo che calcolasse anche le lunghezze delle parole.

Cosi ho deciso di provare a improntare il codice in modo da dare a chi lo esegue la possibbilità di sceglire entrambi i modi creando un menù con 3 opzioni.