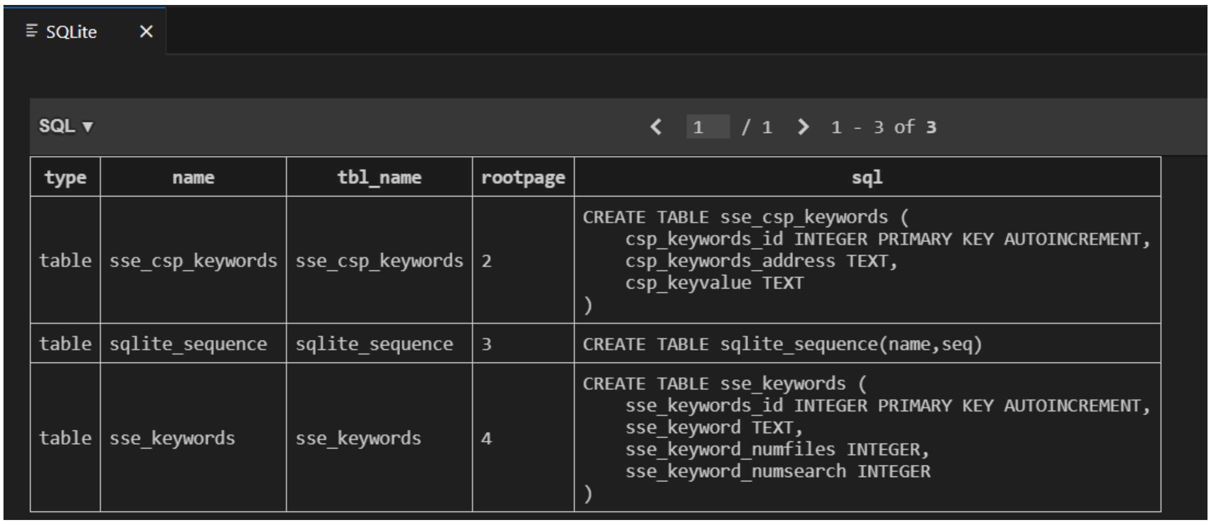
Tutorial 8

I am using sqlite3 with python to set up my tables

Screen shot of both tables

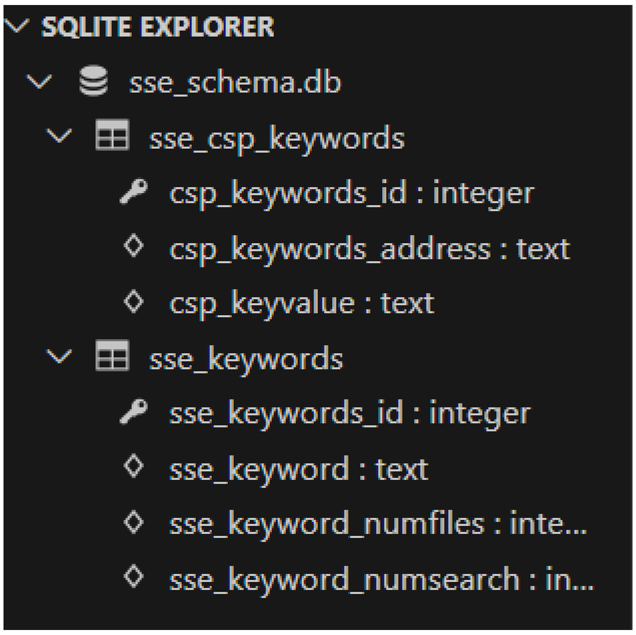


type	name	tbl_name	rootpage	sql

table	sse_csp_keywords	sse_csp_keywords	2	CREATE TABLE sse_csp_keywords (
				    csp_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
				    csp_keywords_address TEXT,
				    csp_keyvalue TEXT
				)

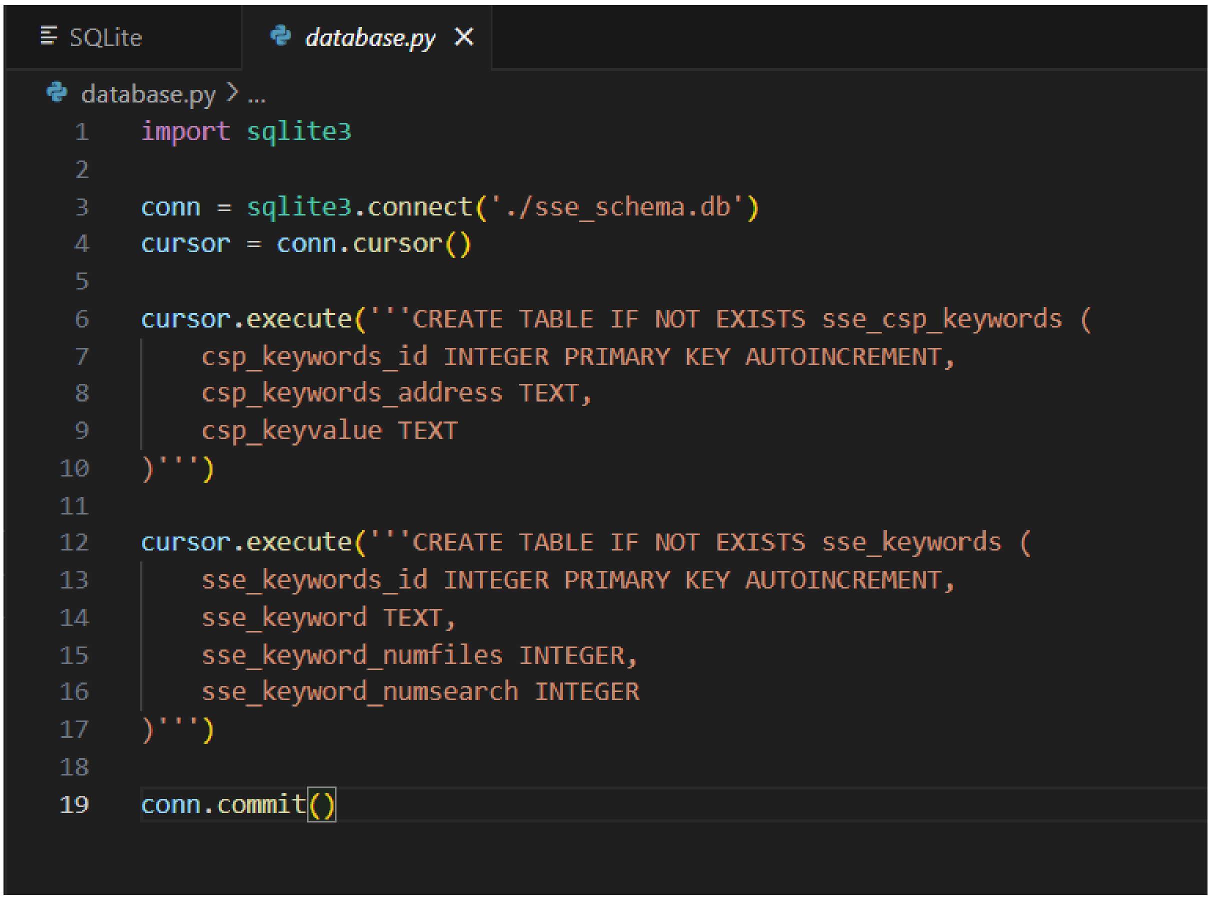
table	sqlite_sequence	sqlite_sequence	3	CREATE TABLE sqlite_sequence(name,seq)

table	sse_keywords	sse_keywords	4	CREATE TABLE sse_keywords (
				    sse_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
				    sse_keyword TEXT,
				    sse_keyword_numfiles INTEGER,
				    sse_keyword_numsearch INTEGER
				)



```
SQLITE EXPLORER
  sse_schema.db
    sse_csp_keywords
      csp_keywords_id : integer
      csp_keywords_address : text
      csp_keyvalue : text
    sse_keywords
      sse_keywords_id : integer
      sse_keyword : text
      sse_keyword_numfiles : inte...
      sse_keyword_numsearch : in...
```

Set up script



```python
import sqlite3

conn = sqlite3.connect('./sse_schema.db')
cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS sse_csp_keywords (
    csp_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
    csp_keywords_address TEXT,
    csp_keyvalue TEXT
)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS sse_keywords (
    sse_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sse_keyword TEXT,
    sse_keyword_numfiles INTEGER,
    sse_keyword_numsearch INTEGER
)''')

conn.commit()
```

Rest of the work by the script

```python
import sqlite3
import os
import hashlib
import time
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import secrets
from cryptography.hazmat.primitives import padding

# Connect to the SQLite database
conn = sqlite3.connect('./sse_schema.db')
cursor = conn.cursor()

# Create necessary tables if they don't exist
cursor.execute('''CREATE TABLE IF NOT EXISTS sse_csp_keywords (
    csp_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
    csp_keywords_address TEXT,
    csp_keyvalue TEXT
)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS sse_keywords (
    sse_keywords_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sse_keyword TEXT,
    sse_keyword_numfiles INTEGER,
    sse_keyword_numsearch INTEGER
)''')

conn.commit()

# Define folder path for text files
folder_path = './textfiles/'

# Define AES key (use your actual KSKE here)
KSKE_hex = '8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91'
KSKE = bytes.fromhex(KSKE_hex)

# AES encryption function with padding
def aes_encrypt(data, key):
    iv = secrets.token_bytes(16)
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
backend=default_backend())
    encryptor = cipher.encryptor()

    # Pad data to ensure it's a multiple of 16 bytes (AES block size)
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()

    return iv + encryptor.update(padded_data) + encryptor.finalize()

# SHA-256 hash function for keywords
def hash_word(word):
    return hashlib.sha256(word.encode()).hexdigest()

# Keyword key computation
def keyword_key(keyword, numsearch):
    combined = f"{keyword}{numsearch}".encode()
    return hashlib.sha256(combined).hexdigest()

# Compute CSP keywords address
def csp_keyword_address(Kw, numfiles):
    combined = f"{Kw}{numfiles}".encode()
    return hashlib.sha256(combined).hexdigest()

# Insert or update keyword in the database
def insert_or_update_keyword(cursor, keyword_hash, numfiles, numsearch):
    # Check if the keyword already exists
    cursor.execute('''
    SELECT sse_keyword_numfiles, sse_keyword_numsearch FROM sse_keywords WHERE
sse_keyword = ?
```

```python
    ''', (keyword_hash,))
    result = cursor.fetchone()

    if result:
        # Keyword exists, update numfiles and numsearch
        existing_numfiles, existing_numsearch = result
        new_numfiles = existing_numfiles + numfiles  # Increment numfiles
        # new_numsearch = existing_numsearch + 1       # Increment numsearch

        cursor.execute('''
        UPDATE sse_keywords SET sse_keyword_numfiles = ?
        WHERE sse_keyword = ?
        ''', (new_numfiles, keyword_hash))
    else:
        # Keyword doesn't exist, insert new record
        cursor.execute('''
        INSERT INTO sse_keywords (sse_keyword, sse_keyword_numfiles,
sse_keyword_numsearch)
        VALUES (?, ?, ?)
        ''', (keyword_hash, numfiles, numsearch))

# Collect all CSP keywords data for later processing
def collect_csp_keywords_data(cursor):
    cursor.execute('''
    SELECT sse_keyword, sse_keyword_numfiles FROM sse_keywords
    ''')
    return cursor.fetchall()

total_size = sum(os.path.getsize(os.path.join(folder_path, f)) for f in
os.listdir(folder_path) if f.endswith('.txt'))

# Start execution timer
start_time = time.time()

# First Pass: Process each file to update the sse_keywords table
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        file_path = os.path.join(folder_path, filename)

        # Read the original file to extract keywords
        with open(file_path, 'r') as file:
            words = file.read().split()

        numfiles = 1  # Each file contains these keywords at least once

        # Process each word to update the sse_keywords table
        for word in words:
            keyword_hash = hash_word(word)
            insert_or_update_keyword(cursor, keyword_hash, numfiles, 1)

# Second Pass: Encrypt and delete all files
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        file_path = os.path.join(folder_path, filename)

        # Read and encrypt the file
        with open(file_path, 'rb') as file:
            data = file.read()
            encrypted_data = aes_encrypt(data, KSKE)

        # Save encrypted file and delete the original
        with open(file_path + '.enc', 'wb') as encrypted_file:
            encrypted_file.write(encrypted_data)

        os.remove(file_path)  # Delete original .txt file

# Third Pass: Compute and insert CSP keywords based on the populated
sse_keywords table
csp_keywords_data = []
for keyword, numfiles in collect_csp_keywords_data(cursor):
    kw = keyword_key(keyword, 1)

    # Print parameters before calculating CSP address
    # print(f"Calculating CSP address with keyword: '{kw}' and numfiles:
{numfiles}")

    address = csp_keyword_address(kw, numfiles)
    encrypted_filename = aes_encrypt(f"{filename}{numfiles}".encode(), KSKE)
    # print(f"Encrypted with aes: {filename} + {numfiles}")
    csp_keywords_data.append((address, encrypted_filename))

# Insert collected CSP keywords data into the database
for address, encrypted_filename in csp_keywords_data:
    cursor.execute('''
    INSERT INTO sse_csp_keywords (csp_keywords_address, csp_keyvalue)
    VALUES (?, ?)
    ''', (address, encrypted_filename))

# Commit all changes
conn.commit()

# End execution timer
end_time = time.time()
execution_time = end_time - start_time

print(f"Total execution time: {execution_time:.2f} seconds")
print(f"Total combined size of test files: {total_size} bytes")

conn.close()
```

I generate 25 random text files as

```
import os
import random

# Folder path for the generated .txt files
folder_path = 'textfiles/'

# Ensure the folder exists
os.makedirs(folder_path, exist_ok=True)

# List of common English words
word_list = [
    "apple", "banana", "orange", "grape", "pineapple", "strawberry",
"blueberry", "mango", "peach", "lemon",
    "cherry", "watermelon", "pear", "plum", "kiwi", "melon", "lime",
"coconut", "apricot", "grapefruit",
    "berry", "papaya", "guava", "fig", "date", "pomegranate", "citrus",
"nectarine", "raspberry", "blackberry"
]

# Function to generate random text content with one unique word per line
def generate_random_text():
    # Choose a random number of unique words for the file (between 5 and 30)
    num_words = random.randint(5, min(30, len(word_list)))  # Ensure it
doesn't exceed the number of unique words
    chosen_words = random.sample(word_list, k=num_words)  # Choose unique
words

    return '\n'.join(chosen_words)  # Join words with line breaks

# Generate 25 .txt files with varied sizes
for i in range(1, 26):
    file_content = generate_random_text()  # Generate unique words for each
file

    file_path = os.path.join(folder_path, f'test_file_{i}.txt')
    with open(file_path, 'w') as f:
        f.write(file_content)

print("25 .txt files with random unique words per line have been created.")
```

It would be much better if 25 txt files are provided since I had to changed this code many times to get possible files.

Output of the script

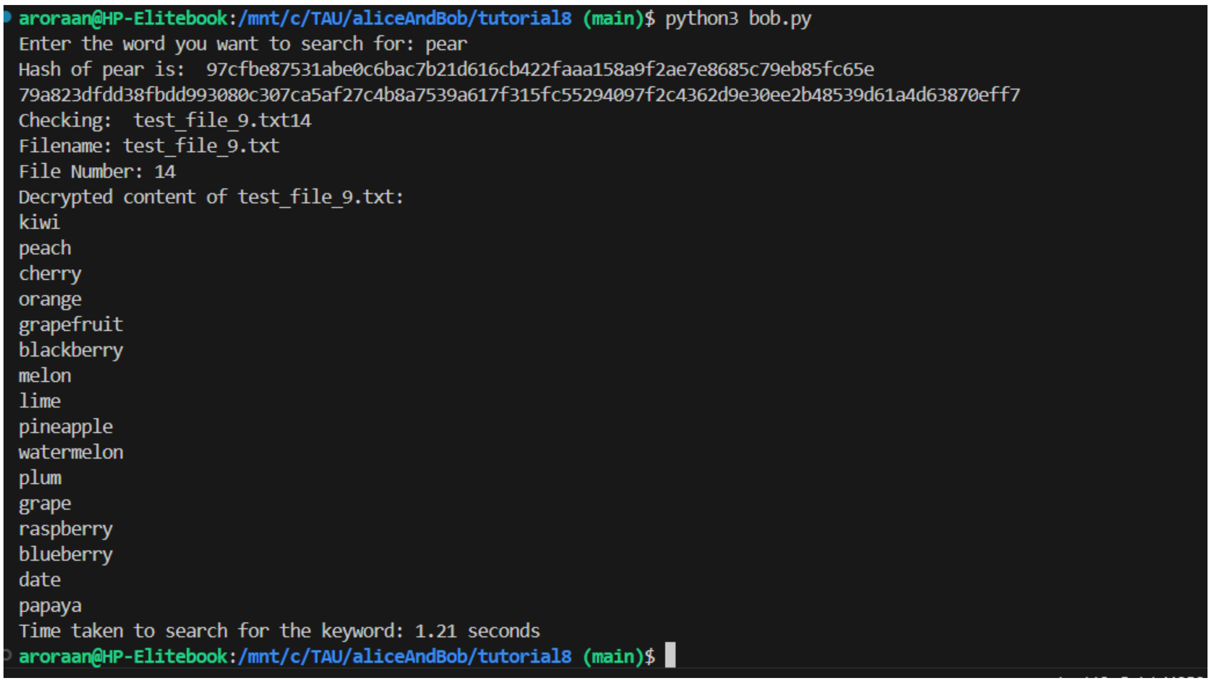
```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$ python3 database.py

KSKE is :  8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91

Total execution time: 8.82 seconds

Total combined size of test files: 2909 bytes
```

Searching script

```python
import hashlib
import time
import sqlite3
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import re

# Define your AES key here (the KSKE from the previous script)
KSKE = '8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91'  #
Replace this with your actual KSKE key from earlier
KSKE = bytes.fromhex(KSKE)

# AES decryption function
def aes_decrypt(data, key):
    iv = data[:16]
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
backend=default_backend())
    decryptor = cipher.decryptor()
    return decryptor.update(data[16:]) + decryptor.finalize()

# SHA-256 hash function for keywords
def hash_word(word):
    print(f"Hash of {word} is: ", hashlib.sha256(word.encode()).hexdigest())
    return hashlib.sha256(word.encode()).hexdigest()

# Keyword key computation
```

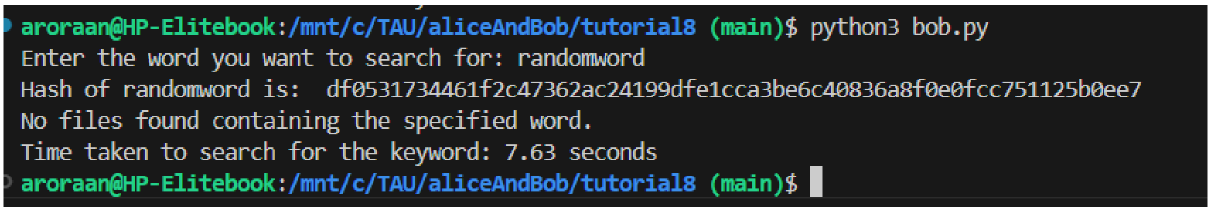
```python
def keyword_key(keyword, numsearch):
    combined = f"{keyword}{numsearch}".encode()
    return hashlib.sha256(combined).hexdigest()

# Compute csp_keywords_address
def csp_keyword_address(Kw, numfiles):
    combined = f"{Kw}{numfiles}".encode()
    return hashlib.sha256(combined).hexdigest()

# Connect to the database
conn = sqlite3.connect('./sse_schema.db')
cursor = conn.cursor()

# Start search timer
start_time = time.time()

# 1. Ask Bob to enter a word to search for
search_word = input("Enter the word you want to search for: ")

# 2. Create SHA-256 hash of the word to generate the keyword value
keyword_hash = hash_word(search_word)

# 3. Retrieve numfiles and numsearch for the keyword from sse_keywords table
cursor.execute('''
SELECT sse_keyword_numfiles, sse_keyword_numsearch FROM sse_keywords WHERE
sse_keyword = ?
''', (keyword_hash,))
result = cursor.fetchone()

if result:
    numfiles, numsearch = result

    # 4. Compute the keyword key Kw
    Kw = keyword_key(keyword_hash, numsearch)
    # print(Kw, numsearch)

    # 5. Compute csp_keywords_address
    csp_address = csp_keyword_address(Kw, numfiles)
    # print(csp_address, numfiles)

    # 6. Retrieve csp_keyvalue using csp_keywords_address
    cursor.execute('''
    SELECT csp_keyvalue FROM sse_csp_keywords WHERE csp_keywords_address = ?
    ''', (csp_address,))
    encrypted_filename_result = cursor.fetchall()

    if encrypted_filename_result:
        # If results are found, decrypt each associated file
        for (encrypted_filename,) in encrypted_filename_result:
            print(encrypted_filename.hex())
            # 7. Decrypt the filename and numfiles from csp_keyvalue
            decrypted_info = aes_decrypt(encrypted_filename, KSKE).decode()
            print("Checking: ", decrypted_info)

            decrypted_info = re.sub(r'\\.*$', '', repr(decrypted_info))
            match = re.match(r"(.+?)(\d+)$", decrypted_info)

            if match:
                filename = match.group(1).lstrip("'")   # Extracts
'test_file_1.txt'
                file_num = match.group(2)   # Extracts '1'
                print("Filename:", filename)
                print("File Number:", file_num)

                # 8. Decrypt the file content
                file_path = f'./textfiles/{filename}.enc'
                try:
                    with open(file_path, 'rb') as encrypted_file:
                        encrypted_data = encrypted_file.read()

                    decrypted_data = aes_decrypt(encrypted_data,
KSKE).decode()
                    print(f"Decrypted content of {filename}:")
                    print(decrypted_data)
                except FileNotFoundError:
                    print(f"File {file_path} not found.")
            else:
                print("Format error in decrypted_info.")
    else:
        print("No CSP key values found for the specified keyword.")

else:
    print("No files found containing the specified word.")

# Measure end time and calculate search time
end_time = time.time()
search_time = end_time - start_time
print(f"Time taken to search for the keyword: {search_time:.2f} seconds")

# Close database connection
conn.close()
```

output



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$ python3 bob.py
Enter the word you want to search for: pear
Hash of pear is:  97cfbe87531abe0c6bac7b21d616cb422faaa158a9f2ae7e8685c79eb85fc65e
79a823dfdd38fbdd993080c307ca5af27c4b8a7539a617f315fc55294097f2c4362d9e30ee2b48539d61a4d63870eff7
Checking:  test_file_9.txt14
Filename: test_file_9.txt
File Number: 14
Decrypted content of test_file_9.txt:
kiwi
peach
cherry
orange
grapefruit
blackberry
melon
lime
pineapple
watermelon
plum
grape
raspberry
blueberry
date
papaya
Time taken to search for the keyword: 1.21 seconds
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$
```

If word not found



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$ python3 bob.py
Enter the word you want to search for: randomword
Hash of randomword is:  df0531734461f2c47362ac24199dfe1cca3be6c40836a8f0e0fcc751125b0ee7
No files found containing the specified word.
Time taken to search for the keyword: 7.63 seconds
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$
```

This assignment would be way easier if txt files were given and csp_keyword updating and when to update numfiles and numsearch was given in more detail. Currently the tutorial takes focus away from actual working of database due to these other issues.