Table of Contents

Preliminaries

I ran into some error and had to find some help so I ran the following code:

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\katchietsnouvaniccur> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
PS C:\Users\katchietsnouvaniccur>

The code you provided is not Python code; it appears to be a PowerShell command.

In PowerShell, the command `Set-ExecutionPolicy` is used to determine the execution policy for scripts on a Windows machine. The execution policy specifies the restrictions placed on running scripts, including PowerShell scripts. The available execution policies are:

1. `Restricted`: *No scripts are allowed to run. This is the default setting and the most secure one.*
2. `AllSigned`: *Only signed scripts (scripts with a trusted digital signature) are allowed to run.*
3. `RemoteSigned`: *Downloaded scripts must be signed, but local scripts can run without a signature.*
4. `Unrestricted`: *All scripts can run without any restrictions. This is the least secure setting.*

The specific command you provided is:

sql Copy code

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

Here, it sets the execution policy for the current PowerShell process to `Bypass`, which means that any scripts or commands can run without any restrictions during the lifetime of this PowerShell session.

The `-Scope Process` parameter specifies that this change only applies to the current PowerShell session and doesn't modify the system-wide execution policy.

It's important to note that changing the execution policy to `Bypass` can pose a security risk, as it allows any script to run without any restrictions. It's generally recommended to use more restrictive policies and only set the execution policy to `Bypass` when you know the source and content of the scripts you are running are safe.

The following code was able to run for pygame

```
# Import the Pygame library, which allows us to create graphical applications and games.
import pygame

# Initialize Pygame. This prepares Pygame for use.
pygame.init()

# Define some colors using RGB values.
white = (255, 255, 255)  # For the background and the color of the snake's head.
black = (0, 0, 0)        # For the outline of the snake's head.
      red = (255, 0, 0)        # Not used in this code, but can be used for other elements.

      # Create a window (display surface) with a resolution of 800x600 pixels, where the game will be drawn.
      dis = pygame.display.set_mode((800, 600))

      # Set the title of the window to "Snake Game".
      pygame.display.set_caption('Snake Game')

      # Initialize a variable to control the main game loop.
      game_over = False

      # Initialize the coordinates of the snake's head (square).
      x1 = 300
      y1 = 300

      # Initialize variables to control the change in position of the snake's head (movement).
      x1_change = 0
      y1_change = 0

      # Create a clock object to control the frame rate of the game.
      clock = pygame.time.Clock()

      # The main game loop.
      while not game_over:
          # Handle events such as user input (e.g., pressing the close button on the window or arrow keys).
          for event in pygame.event.get():
              if event.type == pygame.QUIT:
                  game_over = True
              if event.type == pygame.KEYDOWN:
                  # Check if the arrow keys are pressed and update the movement variables accordingly.
                  if event.key == pygame.K_LEFT:
                      x1_change = -10
                      y1_change = 0
                  elif event.key == pygame.K_RIGHT:
                      x1_change = 10
                      y1_change = 0
                  elif event.key == pygame.K_UP:
                      y1_change = -10
                      x1_change = 0
                  elif event.key == pygame.K_DOWN:
```

```
                    y1_change = 10
                    x1_change = 0

        # Update the position of the snake's head based on the movement variables.
        x1 += x1_change
        y1 += y1_change

        # Clear the screen by filling it with a white color.
        dis.fill(white)

        # Draw a black square representing the snake's head at the updated position.
        pygame.draw.rect(dis, black, [x1, y1, 10, 10])

        # Update the display to show the changes made in the current iteration of the
  loop.
        pygame.display.update()

        # Control the frame rate of the game (30 frames per second).
        clock.tick(30)

    # After the game loop ends (when game_over becomes True):
    # Uninitialize Pygame to clean up resources.
    pygame.quit()

    # Exit the Python program gracefully.
    quit()

```

Music Player

The following code was able to run but not able to play music as the directory was not selected

```
# Import necessary libraries
import os           # For interacting with the operating system
import pygame       # For playing music
import tkinter as tk  # For creating the graphical user interface
from tkinter import filedialog  # For choosing directories and files

# Function to play the selected music
def play_music():
    pygame.mixer.init()  # Initialize the mixer module of pygame
        pygame.mixer.music.load(selected_song.get())  # Load the selected music file
        pygame.mixer.music.play()  # Start playing the music

    # Function to pause the currently playing music
    def pause_music():
        pygame.mixer.music.pause()  # Pause the music

    # Function to unpause the currently paused music
    def unpause_music():
        pygame.mixer.music.unpause()  # Unpause the music

    # Function to stop the currently playing music
    def stop_music():
        pygame.mixer.music.stop()  # Stop the music

    # Function to choose a directory containing music files
    def choose_directory():
        directory = filedialog.askdirectory()  # Open a dialog to select a directory
        os.chdir(directory)  # Change the current working directory to the chosen directory
        song_listbox.delete(0, tk.END)  # Clear the listbox of any previous entries
        for song in os.listdir(directory):  # Loop through files in the directory
            if song.endswith(".mp3"):  # Assuming all songs are in MP3 format
                song_listbox.insert(tk.END, song)  # Insert the song name into the listbox

    # Create the main application window
    root = tk.Tk()
    root.title("Simple Music Player")  # Set the title of the window
    root.geometry("400x300")  # Set the initial size of the window

    selected_song = tk.StringVar()  # Create a variable to store the selected song
    song_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  # Create a listbox to display song names
    song_listbox.pack(pady=20)  # Place the listbox in the window with some padding

    # Create buttons for controlling the music
    play_button = tk.Button(root, text="Play", command=play_music)  # Play button
    play_button.pack()  # Place the button in the window

    pause_button = tk.Button(root, text="Pause", command=pause_music)  # Pause button
    pause_button.pack()  # Place the button in the window

    unpause_button = tk.Button(root, text="Unpause", command=unpause_music)  # Unpause button
```

```
51.      unpause_button.pack()  # Place the button in the window
52.
53.      stop_button = tk.Button(root, text="Stop", command=stop_music)  # Stop button
54.      stop_button.pack()  # Place the button in the window
55.
56.      choose_directory_button = tk.Button(root, text="Choose Directory",
   command=choose_directory)  # Choose directory button
57.      choose_directory_button.pack()  # Place the button in the window
58.
59.      root.mainloop()  # Start the main event loop to display the window and handle
   user interactions
60.
```

Trial Error for music app to open a directory and play music

The following code successfully opened 2 popups, one showing the music player, another one showing the hard coded music directory path, but failed to play music.

```
1. # Import necessary libraries
2. import os           # For interacting with the operating system
3. import pygame       # For playing music
4. import tkinter as tk  # For creating the graphical user interface
5. from tkinter import filedialog  # For choosing directories and files
6.
7. # Function to play the selected music
8. def play_music():
9.     pygame.mixer.init()  # Initialize the mixer module of pygame
10.         pygame.mixer.music.load(selected_song.get())  # Load the selected music file
11.         pygame.mixer.music.play()  # Start playing the music
12.
13.     # Function to pause the currently playing music
14.     def pause_music():
15.         pygame.mixer.music.pause()  # Pause the music
16.
17.     # Function to unpause the currently paused music
18.     def unpause_music():
19.         pygame.mixer.music.unpause()  # Unpause the music
20.
21.     # Function to stop the currently playing music
22.     def stop_music():
23.         pygame.mixer.music.stop()  # Stop the music
24.
25.     # Function to load music from a specific directory
26.     def load_music_from_directory(directory_path):
27.         os.chdir(directory_path)  # Change the current working directory to the
    chosen directory
28.         song_listbox.delete(0, tk.END)  # Clear the listbox of any previous entries
29.         for song in os.listdir(directory_path):  # Loop through files in the
    directory
30.             if song.endswith(".mp3"):  # Assuming all songs are in MP3 format
31.                 song_listbox.insert(tk.END, song)  # Insert the song name into the
    listbox
32.         if song_listbox.size() > 0:  # If there are songs in the listbox
33.             selected_song.set(os.path.join(directory_path, song_listbox.get(0)))  #
    Set the selected song to the first song in the list
34.
35.     # Create the main application window
36.     root = tk.Tk()
37.     root.title("Simple Music Player")  # Set the title of the window
38.     root.geometry("400x300")  # Set the initial size of the window
39.
40.     selected_song = tk.StringVar()  # Create a variable to store the selected song
41.     song_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  # Create a listbox to
    display song names
42.     song_listbox.pack(pady=20)  # Place the listbox in the window with some padding
43.
44.     # Load music from a specific directory (update this path to your music directory)
45.     load_music_from_directory(r"D:\Documents\Fl new space\++ Rendered Here")
46.
47.     # Function to choose a directory containing music files
48.     def choose_directory():
49.         directory = filedialog.askdirectory()  # Open a dialog to select a directory
```

```
50.          load_music_from_directory(directory)  # Load the music from the chosen
   directory
51.
52.      # Function to handle the selection of a song from the listbox
53.      def on_song_selected(event):
54.          selected_song_index = song_listbox.curselection()
55.          if selected_song_index:
56.              selected_song_name = song_listbox.get(selected_song_index[0])
57.              selected_song_path = os.path.join(directory_path.get(),
   selected_song_name)
58.              selected_song.set(selected_song_path)
59.
60.      # ... (rest of the code remains the same)
61.
62.      # Bind the on_song_selected function to the <<ListboxSelect>> event
63.      song_listbox.bind("<<ListboxSelect>>", on_song_selected)
64.
65.      # Create the main application window
66.      root = tk.Tk()
67.      root.title("Simple Music Player")  # Set the title of the window
68.      root.geometry("400x300")  # Set the initial size of the window
69.
70.      selected_song = tk.StringVar()  # Create a variable to store the selected song
71.      song_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  # Create a listbox to
   display song names
72.      song_listbox.pack(pady=20)  # Place the listbox in the window with some padding
73.
74.      # Create buttons for controlling the music
75.      play_button = tk.Button(root, text="Play", command=play_music)  # Play button
76.      play_button.pack()  # Place the button in the window
77.
78.      pause_button = tk.Button(root, text="Pause", command=pause_music)  # Pause button
79.      pause_button.pack()  # Place the button in the window
80.
81.      unpause_button = tk.Button(root, text="Unpause", command=unpause_music)  #
   Unpause button
82.      unpause_button.pack()  # Place the button in the window
83.
84.      stop_button = tk.Button(root, text="Stop", command=stop_music)  # Stop button
85.      stop_button.pack()  # Place the button in the window
86.
87.      choose_directory_button = tk.Button(root, text="Choose Directory",
   command=choose_directory)  # Choose directory button
88.      choose_directory_button.pack()  # Place the button in the window
89.
90.      root.mainloop()  # Start the main event loop to display the window and handle
   user interactions
91.
92.
```

1st Working music App

The following code was able to play music automatically

```
1. # Import necessary libraries
2. import os           # For interacting with the operating system
3. import pygame       # For playing music
4. import tkinter as tk  # For creating the graphical user interface
5.
6. # Function to play the selected music
7. def play_music():
8.     pygame.mixer.init()  # Initialize the mixer module of pygame
9.     pygame.mixer.music.load(selected_song.get())  # Load the selected music file
10.          pygame.mixer.music.play()  # Start playing the music
11.
12.      # Function to pause the currently playing music
13.      def pause_music():
14.          pygame.mixer.music.pause()  # Pause the music
15.
16.      # Function to unpause the currently paused music
17.      def unpause_music():
18.          pygame.mixer.music.unpause()  # Unpause the music
19.
20.      # Function to stop the currently playing music
21.      def stop_music():
22.          pygame.mixer.music.stop()  # Stop the music
23.
24.      # Create the main application window
25.      root = tk.Tk()
26.      root.title("Simple Music Player")  # Set the title of the window
27.      root.geometry("400x300")  # Set the initial size of the window
28.
29.      selected_song = tk.StringVar()  # Create a variable to store the selected song
30.      selected_song.set(r"D:\Documents\Fl new space\++ Rendered Here\FunEDM.mp3")  #
   Set the selected song to "FunEDM.mp3"
31.
32.      # Function to load music from a specific directory
33.      def load_music_from_directory(directory_path):
34.          pass  # Since we don't need to load music from the directory, we simply pass
35.
36.      # ... (rest of the code remains the same)
37.
38.      # Create the main application window
39.      root = tk.Tk()
40.      root.title("Simple Music Player")  # Set the title of the window
41.      root.geometry("400x300")  # Set the initial size of the window
42.
43.      selected_song = tk.StringVar()  # Create a variable to store the selected song
44.      selected_song.set(r"D:\Documents\Fl new space\++ Rendered Here\Real EDM_8_1.Real
   EDM_8_1.mp3")  # Set the selected song to "Real EDM_8_1.mp3"
45.
46.      # ... (rest of the code remains the same)
47.
48.      # Create buttons for controlling the music
49.      play_button = tk.Button(root, text="Play", command=play_music)  # Play button
50.      play_button.pack()  # Place the button in the window
51.
52.      pause_button = tk.Button(root, text="Pause", command=pause_music)  # Pause button
```

```
53.      pause_button.pack()  # Place the button in the window
54.
55.      unpause_button = tk.Button(root, text="Unpause", command=unpause_music)  #
   Unpause button
56.      unpause_button.pack()  # Place the button in the window
57.
58.      stop_button = tk.Button(root, text="Stop", command=stop_music)  # Stop button
59.      stop_button.pack()  # Place the button in the window
60.
61.      root.mainloop()  # Start the main event loop to display the window and handle
   user interactions
```

Generating .exe file

I was able to generate an executable file by running the following in the terminal:

PS C:\Users\katchietsnouvaniccur\music_player_project> pip install pygame
PS C:\Users\katchietsnouvaniccur\music_player_project> pip install pyinstalleR
PS C:\Users\katchietsnouvaniccur\music_player_project> cd D:\Documents\Python_Projects\MusicPlayer
PS D:\Documents\Python_Projects\MusicPlayer> pyinstaller MusicPlayer2.py --onefile

The process of converting a Python script into a standalone .exe file using tools like PyInstaller or cx_Freeze can be convenient for distributing simple applications or scripts. However, it may not be the best solution for all scenarios, especially for complex or production-level programs. Here are some considerations to keep in mind:

1. Platform Dependency: When you convert a Python script to a .exe file, the resulting executable is specific to the operating system on which it was created. For example, an .exe file generated on Windows will only run on Windows. If you need to distribute your application across multiple platforms (Windows, macOS, Linux), you'll have to create separate executables for each platform.
2. Size and Overhead: Standalone .exe files tend to be larger in size compared to the original Python script due to bundling all the necessary dependencies with the executable. This can lead to increased overhead, especially if your application has many dependencies or resources.
3. Dependency Management: With a standalone .exe file, all dependencies are packaged into a single file, which may lead to potential version conflicts or difficulties in updating individual components.
4. Limited Debugging: Debugging issues in the standalone .exe file can be more challenging compared to debugging the original Python script directly in the Python environment.
5. Security Considerations: Standalone .exe files may be susceptible to reverse engineering, and in some cases, antivirus software may flag them as potentially harmful due to their packaging nature.
6. Licensing and Compliance: If your Python script or application uses third-party libraries, you need to ensure compliance with their licensing terms when distributing the application.

For production-level programs, other deployment methods may be more suitable, depending on the complexity and requirements of the project:

1. Virtual Environments: Using virtual environments allows you to manage dependencies more effectively and provides better isolation for different projects.
2. Packaging with Wheels: You can package your Python project into a wheel distribution, making it easier to distribute and install through pip.
3. Packaging Systems: For more complex applications, using packaging systems like conda or setuptools can provide more flexibility and customization in deploying Python applications.
4. Containers: Containerization tools like Docker can help create portable and reproducible environments for your applications, ensuring consistency across different systems.
5. Remember that the choice of deployment method depends on the specific needs and requirements of your project. For small, simple scripts, generating .exe files might be sufficient, but for larger and more complex applications, it's essential to consider more robust and scalable deployment strategies.

Converting File Snffer.c file to FileSNiffer.py file

I was able to convert the C language code to python language code. This was due to the difficulity in making a GUI for C.

```
# from ctypes import windll
# from pathlib import Path

# Import required modules
import os
import ctypes

# Function to list directories and files in the specified path
def list_directories_and_files_in_root(path, log_file):
        # Write a heading for directories in the log file
        log_file.write("Directories:\n")
        # Loop through all entries (files and directories) in the specified path
        for entry in os.listdir(path):
            # Check if the entry is a directory
            if os.path.isdir(os.path.join(path, entry)):
                # Exclude special directories '.' and '..' from the listing
                if entry not in ('.', '..'):
                    # Print the directory name to the console
                    print(f"Directory: {entry}")
                    # Write the directory name to the log file
                    log_file.write(f"Directory: {entry}\n")

        # Write a heading for files in the log file
        log_file.write("\nFiles:\n")
        # Loop through all entries (files and directories) in the specified path
    again
        for entry in os.listdir(path):
            # Check if the entry is a file
            if os.path.isfile(os.path.join(path, entry)):
                # Print the file name to the console
                print(f"File: {entry}")
                # Write the file name to the log file
                log_file.write(f"File: {entry}\n")

    # Function to get the path of a special folder using CSIDL
    def get_folder_path(csidl):
        # Create a buffer to store the folder path (use larger buffer size to
    accommodate longer paths)
        buf = ctypes.create_unicode_buffer(1024)  # 1024 characters buffer size
        # Call the Windows API function to get the folder path using CSIDL
        ctypes.windll.shell32.SHGetFolderPathW(None, csidl, None, 0, buf)
        return buf.value

    # Function to process default special folders
    def process_default_folders(log_file):
        # Define an array of folder CSIDLs (constants) along with their descriptions
        default_folder_constants = [
            (0x0000, "This is the Desktop folder:"),
            (0x0005, "This is the Documents folder:"),
            (0x000E, "This is the Videos folder:"),
            (0x0027, "This is the Pictures folder:"),
            (0x000D, "This is the Music folder:"),
            (0x0019, "This is the Common Desktop folder:")
        ]
```

```
53.
54.          # Loop through the default folders and get their paths using CSIDL
55.          for csidl, description in default_folder_constants:
56.              try:
57.                  # Get the path of the default folder using CSIDL
58.                  folder_path = get_folder_path(csidl)
59.                  # Write the folder description to the log file
60.                  log_file.write(f"\n\n{description}\n\n")
61.                  # Call the function to list directories and files in the specified
   folder
62.                  list_directories_and_files_in_root(folder_path, log_file)
63.              except Exception as e:
64.                  # Print an error message if getting the folder path fails
65.                  print(f"Failed to get default folder path for CSIDL {csidl}. Error:
   {e}")
66.
67.      # Main function
68.      def main():
69.          # Get the path of the 'Documents' folder
70.          documents_folder_path = os.path.expanduser("~/Documents")
71.
72.          # Print the current directory and the code file's location
73.          print(f"Listing directories and files in {documents_folder_path}")
74.          current_code_file_path = os.path.abspath(__file__)
75.          print(f"Current Code File Location: {current_code_file_path}")
76.
77.          # Create the log file name using the current directory path
78.          log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin",
   "1.txt")
79.
80.          # Check if the 'bin' folder exists, if not, create it
81.          bin_folder_path = os.path.join(os.path.dirname(current_code_file_path),
   "bin")
82.          if not os.path.exists(bin_folder_path):
83.              os.makedirs(bin_folder_path)
84.
85.          # Open the log file in "append" mode (add data to the existing file)
86.          with open(log_file_name, "a") as log_file:
87.              # Call the function to list directories and files in the specified path
   (Documents folder)
88.              list_directories_and_files_in_root(documents_folder_path, log_file)
89.
90.              # Call the new function to iterate through the default folders and append
   data to the log file
91.              process_default_folders(log_file)
92.
93.          # Print a success message with the log file name
94.          print(f"Successfully logged to {log_file_name}")
95.
96.          # Read and print the contents of the log file to the console
97.          print("\nLogged Contents:")
98.          with open(log_file_name, "r") as read_log_file:
99.              for line in read_log_file:
100.                 print(line.rstrip())
101.
102.     # Execute the main function if this script is run directly (not imported as a
   module)
```

```
    if __name__ == "__main__":
        main()

```

Adding multiple default folder

```
# Function to process default special folders
def process_default_folders(log_file):
    # Define an array of folder CSIDLs (constants) along with their descriptions
    all_csidl_values = [
        (0x0000, "CSIDL_DESKTOP", "The virtual folder representing the Windows desktop, the root of the shell namespace."),
        (0x0005, "CSIDL_DOCUMENTS", "This is the Documents folder:"),
        (0x000E, "CSIDL_VIDEOS", "This is the Videos folder:"),
        (0x0027, "CSIDL_PICTURES", "This is the Pictures folder:"),
        (0x000D, "CSIDL_MUSIC", "This is the Music folder:"),
            (0x0019, "CSIDL_COMMON_DESKTOPDIRECTORY", "This is the Common Desktop folder:"),
            (0x0002, "CSIDL_PROGRAMS", "The file system directory that contains the user's program groups (which are themselves file system directories)"),
            (0x11, "CSIDL_DRIVES", ""),
            (0x22, "CSIDL_HISTORY", ""),
            (0x20, "CSIDL_INTERNET_CACHE", ""),
            (0x12, "CSIDL_NETWORK", ""),
            (0x08, "CSIDL_RECENT", "The file system directory that contains shortcuts to the user's most recently used documents"),
            (0x02, "CSIDL_PROGRAMS", ""),
            (0x2C, "CSIDL_PROGRAM_FILES_COMMONX86", "A folder for 32-bit components that are shared across applications on 64-bit systems"),
            (0x2B, "CSIDL_PROGRAM_FILES_COMMON", "A folder for components that are shared across applications"),
            (0x2A, "CSIDL_PROGRAM_FILESX86", "The Program Files folder for 32-bit programs on 64-bit systems"),
            (0x26, "CSIDL_PROGRAM_FILES", "The Program Files folder.")
        ]
```

Trial And Error Adding Downloads Folder

Accessing Downloads folder using new method

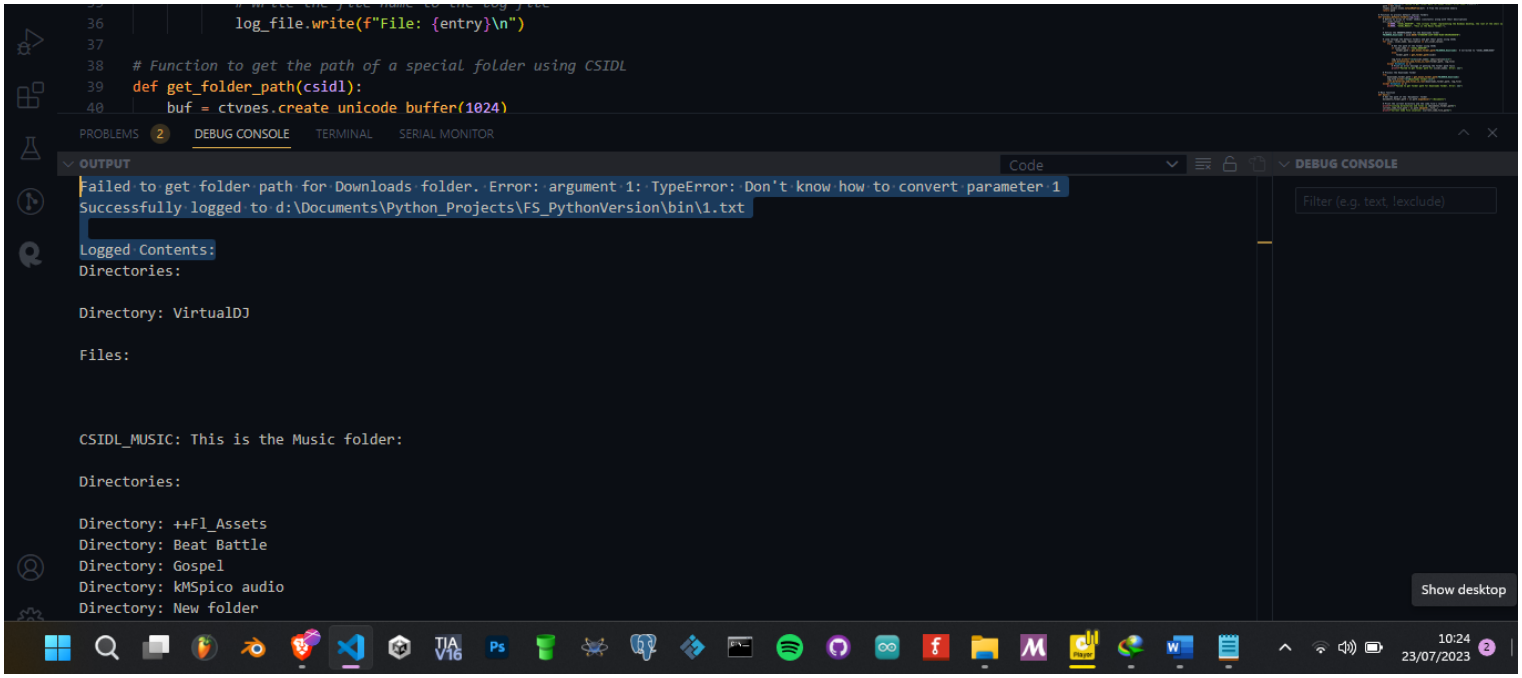
To get the Downloads folder, you have to be running on Vista or newer. The CSIDL values are not available for the Downloads folder on Windows versions older than Vista SHGetFolderPath API. Instead, the Downloads folder can be accessed using the newer SHGetKnownFolderPath API. If you want to maintain some semblance of 'compatibility' on older releases of Windows, you can use the default value listed in the KNOWNFOLDERID page if the API is not available to you. You can use run-time linking to use the API to allow the application to run on the older release of Windows.

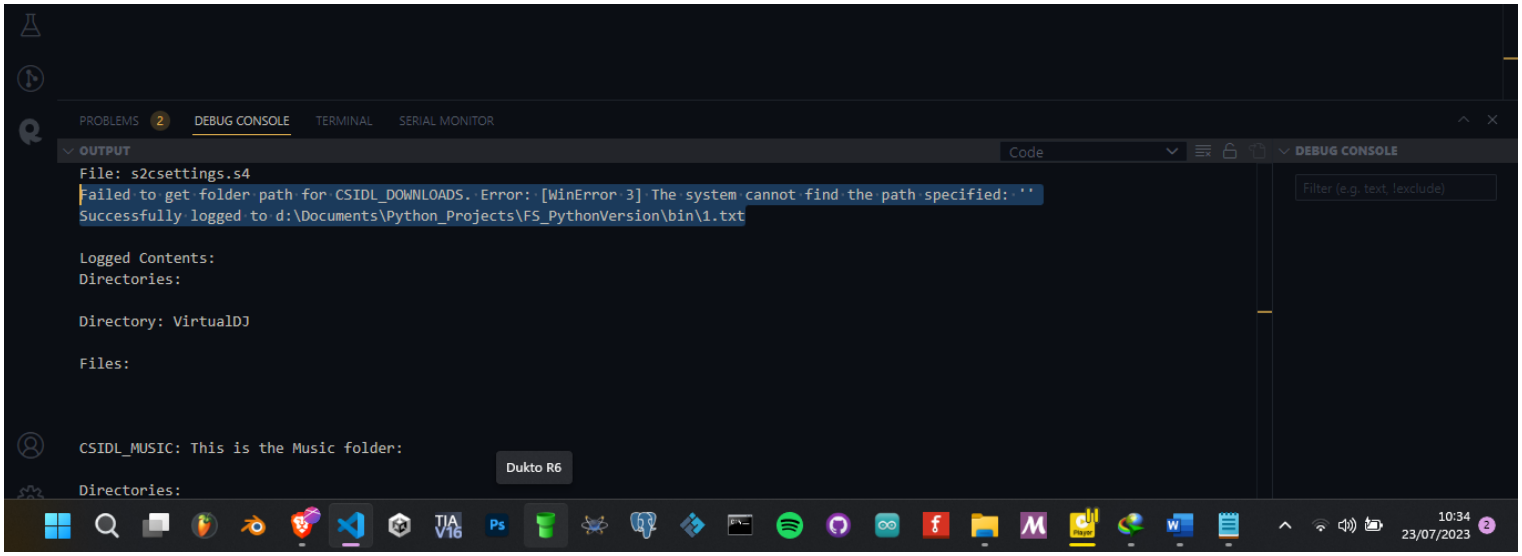
The following code was able to run but couldn't retrieve the downloads folder so **this version will raise an exception if there's an error while fetching the path for the Known Folder.**

```
# import os
# from ctypes import windll
# import ctypes
# from pathlib import Path

# Import required modules
import os
import ctypes
import uuid
    import ctypes.wintypes as wintypes

    # Function to list directories and files in the specified path
    def list_directories_and_files_in_root(path, log_file):
        # Write a heading for directories in the log file
        log_file.write("Directories:\n\n")
        # Loop through all entries (files and directories) in the specified path
        for entry in os.listdir(path):
            # Check if the entry is a directory
            if os.path.isdir(os.path.join(path, entry)):
                # Exclude special directories '.' and '..' from the listing
                if entry not in ('.', '..'):
                    # Print the directory name to the console
                    print(f"Directory: {entry}")
                    # Write the directory name to the log file
                    log_file.write(f"Directory: {entry}\n")

        # Write a heading for files in the log file
        log_file.write("\nFiles:\n\n")
        # Loop through all entries (files and directories) in the specified path again
        for entry in os.listdir(path):
            # Check if the entry is a file
            if os.path.isfile(os.path.join(path, entry)):
                # Print the file name to the console
                print(f"File: {entry}")
                # Write the file name to the log file
                log_file.write(f"File: {entry}\n")

    # Function to get the path of a special folder using CSIDL
    def get_folder_path(csidl):
        buf = ctypes.create_unicode_buffer(1024)
        ctypes.windll.shell32.SHGetFolderPathW(None, csidl, None, 0, buf)
        return buf.value

    # Function to get the path of a known folder using KNOWNFOLDERID
    def get_known_folder_path(known_folder_id):
        buf = ctypes.c_wchar_p()
```

```
47.          result = ctypes.windll.shell32.SHGetKnownFolderPath(known_folder_id, 0, None,
   ctypes.byref(buf))
48.          if result != 0:
49.              raise OSError(f"Failed to get folder path for Known Folder. Error code:
   {result}")
50.          path = buf.value
51.          ctypes.windll.ole32.CoTaskMemFree(buf)  # Free the allocated memory
52.          return path
53.
54.      # Function to process default special folders
55.      def process_default_folders(log_file):
56.          # Define an array of folder CSIDLs (constants) along with their descriptions
57.          all_csidl_values = [
58.              (0x0000, "CSIDL_DESKTOP", "The virtual folder representing the Windows
   desktop, the root of the shell namespace."),
59.              (0x000D, "CSIDL_MUSIC", "This is the Music folder:"),
60.          ]
61.
62.          # Define the KNOWNFOLDERID for the Downloads folder
63.          FOLDERID_Downloads = uuid.UUID("374DE290-123F-4565-9164-39C4925E467B")
64.
65.          # Loop through the default folders and get their paths using CSIDL
66.          for csidl, csidl_name, description in all_csidl_values:
67.              try:
68.                  # Get the path of the folder using CSIDL
69.                  if csidl_name == "CSIDL_DESKTOP":
70.                      folder_path = get_known_folder_path(FOLDERID_Downloads)  #
   Corrected to "CSIDL_DOWNLOADS"
71.                  else:
72.                      folder_path = get_folder_path(csidl)
73.
74.                  log_file.write(f"\n\n{csidl_name}: {description}\n\n")
75.                  list_directories_and_files_in_root(folder_path, log_file)
76.              except Exception as e:
77.                  # Print an error message if getting the folder path fails
78.                  print(f"Failed to get folder path for {csidl_name}. Error: {e}")
79.
80.          # Process the Downloads folder
81.          try:
82.              downloads_folder_path = get_known_folder_path(FOLDERID_Downloads)
83.              log_file.write("\n\nDownloads folder:\n\n")
84.              list_directories_and_files_in_root(downloads_folder_path, log_file)
85.          except Exception as e:
86.              print(f"Failed to get folder path for Downloads folder. Error: {e}")
87.
88.      # Main function
89.      def main():
90.          # Get the path of the 'Documents' folder
91.          documents_folder_path = os.path.expanduser("~/Documents")
92.
93.          # Print the current directory and the code file's location
94.          print(f"Listing directories and files in {documents_folder_path}")
95.          current_code_file_path = os.path.abspath(__file__)
96.          print(f"Current Code File Location: {current_code_file_path}")
97.
98.          # Create the log file name using the current directory path
```

```
99.          log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin",
   "1.txt")
100.
101.         # Check if the 'bin' folder exists, if not, create it
102.         bin_folder_path = os.path.join(os.path.dirname(current_code_file_path),
   "bin")
103.         if not os.path.exists(bin_folder_path):
104.             os.makedirs(bin_folder_path)
105.
106.         # Open the log file in "append" mode (add data to the existing file)
107.         with open(log_file_name, "a") as log_file:
108.             # Call the function to list directories and files in the specified path
   (Documents folder)
109.             list_directories_and_files_in_root(documents_folder_path, log_file)
110.
111.             # Call the new function to iterate through the default folders and append
   data to the log file
112.             process_default_folders(log_file)
113.
114.         # Print a success message with the log file name
115.         print(f"Successfully logged to {log_file_name}")
116.
117.         # Read and print the contents of the log file to the console
118.         print("\nLogged Contents:")
119.         with open(log_file_name, "r") as read_log_file:
120.             for line in read_log_file:
121.                 print(line.rstrip())
122.
123.     if __name__ == "__main__":
124.         main()
125.
```




PROBLEMS 2
DEBUG CONSOLE
TERMINAL
SERIAL MONITOR
OUTPUT
Code
DEBUG CONSOLE
Filter (e.g. text, !exclude)
File: s2csettings.s4
Failed to get folder path for CSIDL_DOWNLOADS. Error: [WinError 3] The system cannot find the path specified: ''
Successfully logged to d:\Documents\Python_Projects\FS_PythonVersion\bin\1.txt
Logged Contents:
Directories:
Directory: VirtualDJ
Files:
CSIDL_MUSIC: This is the Music folder:
Directories:
Dukto R6
10:34
23/07/2023

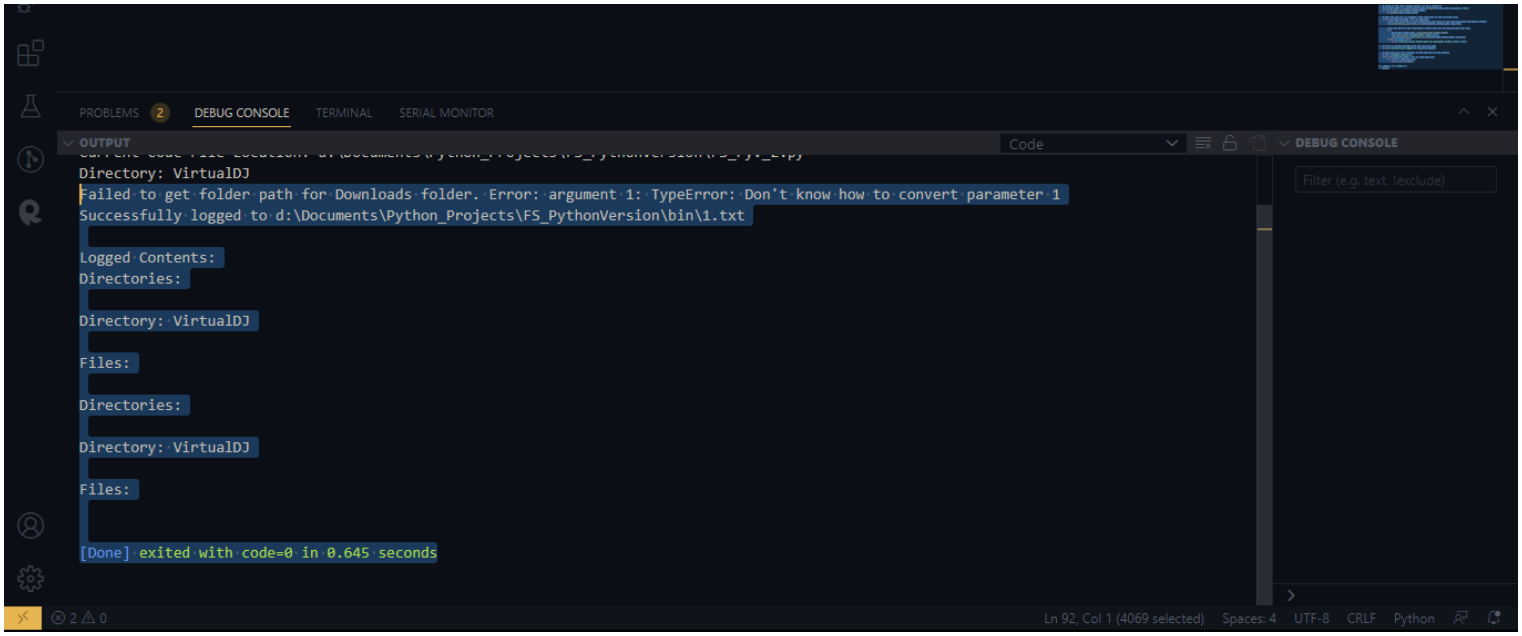
Accessing multiple operating system Downloads folders

If the following could have worked, it would have been benefitial im accessing for multiple operationg systems:

```
# Import required modules
import os
import ctypes
import ctypes.wintypes as wintypes
import uuid
from pathlib import Path

# Function to list directories and files in the specified path
def list_directories_and_files_in_root(path, log_file):
          # Write a heading for directories in the log file
          log_file.write("Directories:\n\n")
          # Loop through all entries (files and directories) in the specified path
          for entry in os.listdir(path):
              # Check if the entry is a directory
              if os.path.isdir(os.path.join(path, entry)):
                  # Exclude special directories '.' and '..' from the listing
                  if entry not in ('.', '..'):
                      # Print the directory name to the console
                      print(f"Directory: {entry}")
                      # Write the directory name to the log file
                      log_file.write(f"Directory: {entry}\n")

          # Write a heading for files in the log file
          log_file.write("\nFiles:\n\n")
          # Loop through all entries (files and directories) in the specified path
   again
          for entry in os.listdir(path):
              # Check if the entry is a file
              if os.path.isfile(os.path.join(path, entry)):
                  # Print the file name to the console
                  print(f"File: {entry}")
                  # Write the file name to the log file
                  log_file.write(f"File: {entry}\n")

      # Function to get the path of the "Downloads" folder
      def get_downloads_folder_path():
          if os.name == 'nt':
              # Windows system, use Windows API to get the "Downloads" folder path
              FOLDERID_Downloads = uuid.UUID("374DE290-123F-4565-9164-39C4925E467B")
              buf = ctypes.c_wchar_p()
              result = ctypes.windll.shell32.SHGetKnownFolderPath(FOLDERID_Downloads,
   0, None, ctypes.byref(buf))
              if result != 0:
                  raise OSError(f"Failed to get folder path for Downloads folder. Error
   code: {result}")
              path = buf.value
              ctypes.windll.ole32.CoTaskMemFree(buf)  # Free the allocated memory
              return path
          else:
              # Non-Windows system, use pathlib to get the "Downloads" folder path
              return str(Path.home() / "Downloads")

      # Main function
```

```
51.      def main():
52.          # Get the path of the 'Documents' folder
53.          documents_folder_path = os.path.expanduser("~/Documents")
54.
55.          # Print the current directory and the code file's location
56.          print(f"Listing directories and files in {documents_folder_path}")
57.          current_code_file_path = os.path.abspath(__file__)
58.          print(f"Current Code File Location: {current_code_file_path}")
59.
60.          # Create the log file name using the current directory path
61.          log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin",
   "1.txt")
62.
63.          # Check if the 'bin' folder exists, if not, create it
64.          bin_folder_path = os.path.join(os.path.dirname(current_code_file_path),
   "bin")
65.          if not os.path.exists(bin_folder_path):
66.              os.makedirs(bin_folder_path)
67.
68.          # Open the log file in "append" mode (add data to the existing file)
69.          with open(log_file_name, "a") as log_file:
70.              # Call the function to list directories and files in the specified path
   (Documents folder)
71.              list_directories_and_files_in_root(documents_folder_path, log_file)
72.
73.              # Get the path of the "Downloads" folder and list its directories and
   files
74.              try:
75.                  downloads_folder_path = get_downloads_folder_path()
76.                  log_file.write("\n\nDownloads folder:\n\n")
77.                  list_directories_and_files_in_root(downloads_folder_path, log_file)
78.              except Exception as e:
79.                  print(f"Failed to get folder path for Downloads folder. Error: {e}")
80.
81.          # Print a success message with the log file name
82.          print(f"Successfully logged to {log_file_name}")
83.
84.          # Read and print the contents of the log file to the console
85.          print("\nLogged Contents:")
86.          with open(log_file_name, "r") as read_log_file:
87.              for line in read_log_file:
88.                  print(line.rstrip())
89.
90.      if __name__ == "__main__":
91.          main()
92.
```

But it brought the following error:


PROBLEMS 2
DEBUG CONSOLE
TERMINAL
SERIAL MONITOR
OUTPUT
Code
DEBUG CONSOLE
Filter (e.g. text, !exclude)
Directory: VirtualDJ
Failed to get folder path for Downloads folder. Error: argument 1: TypeError: Don't know how to convert parameter 1
Successfully logged to d:\Documents\Python_Projects\FS_PythonVersion\bin\1.txt
Logged Contents:
Directories:
Directory: VirtualDJ
Files:
Directories:
Directory: VirtualDJ
Files:
[Done] exited with code=0 in 0.645 seconds
Ln 92, Col 1 (4069 selected)
Spaces: 4
UTF-8
CRLF
Python

Getting user's profile name and adding path to the gotten path

Try 1

```
1. # Function to get the path of the "Downloads" folder
2. def get_downloads_folder_path():
3.     if os.name == 'nt':
4.         # Windows system, try to get the "Downloads" folder from environment variables
5.         downloads_folder_path = os.getenv('USERPROFILE')
6.         if downloads_folder_path:
7.             return os.path.join(downloads_folder_path, 'Downloads')
8.     else:
9.         # Non-Windows system, use pathlib to get the "Downloads" folder path
10.             return str(Path.home() / "Downloads")
11.
12.         raise OSError("Failed to get folder path for Downloads folder.")
```

Result:

```
'C:\\Users\\katchietsnouvaniccur\\Downloads'
```

The code would have succeeeded in finding the downloads folder and displaying its contents had it returned the following path:
Home\Downloads or
C:\Users\katchietsnouvaniccur\Downloads
but you see, it returned :
C:\\Users\\katchietsnouvaniccur\\Downloads

The path **returned with double backslashes** (C:\Users\katchietsnouvaniccur\Downloads) is due to the way **Python escapes backslashes in string literals.** However, this does not affect the functionality of the code. Python **uses** double backslashes to **represent a single backslash in a string literal**, but internally, the path is still treated correctly.
When you use the path for file operations or any other functions that require a valid file path, the double backslashes will be automatically handled by Python, and it will work as expected.
For example, if you use the path to open a file, you can do it like this:

```
1. file_path = "C:\\Users\\katchietsnouvaniccur\\Downloads\\example.txt"
2. with open(file_path, "r") as file:
3.     content = file.read()
```

Python will interpret the path correctly when opening the file, and it will work as intended.So, you don't need to worry about the double backslashes in the returned path; it will function correctly in Python for file operations and other functions that require a valid file path.

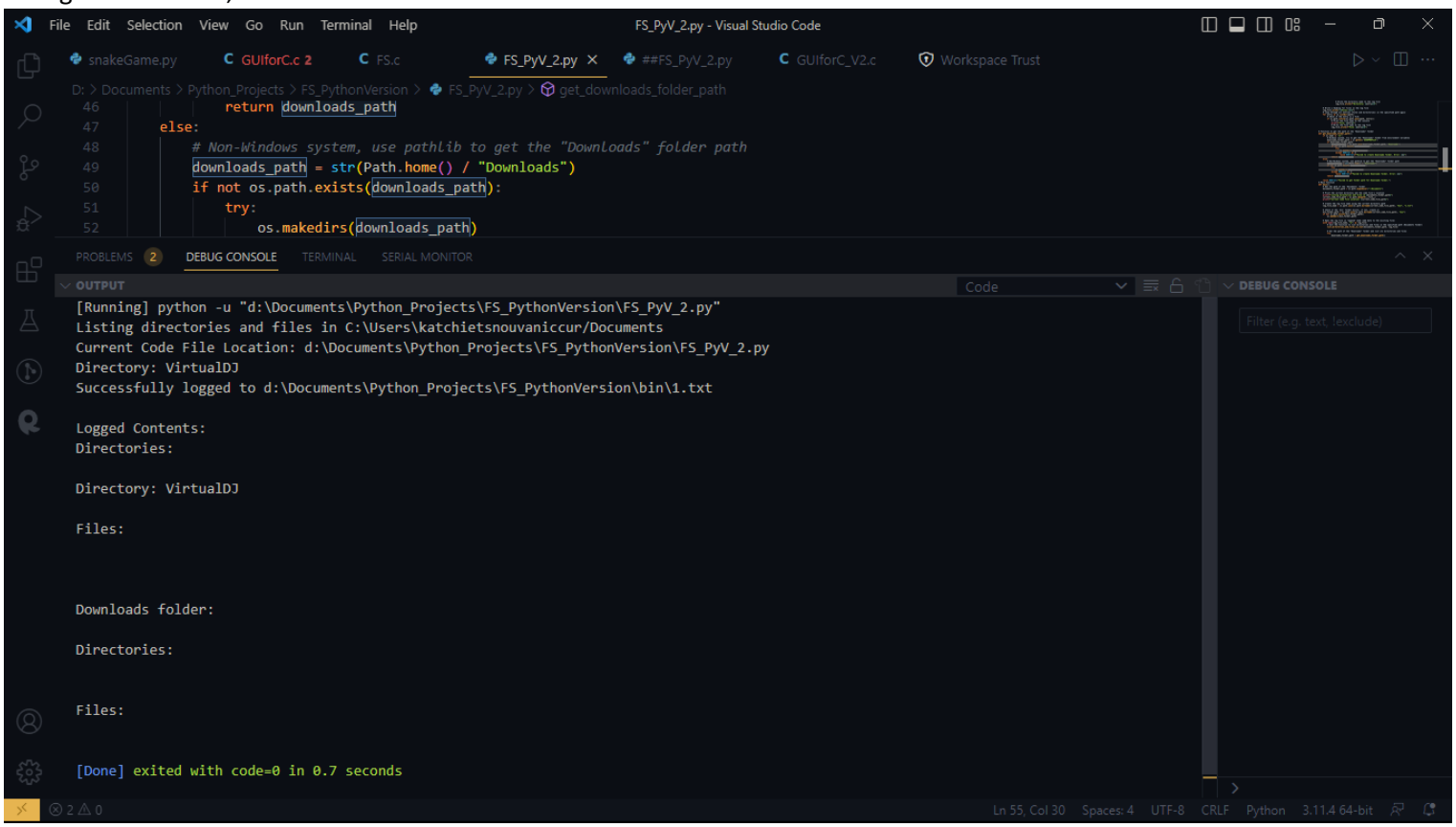
Try 2

The following tracked the old Downloads location:

```
1. # Function to get the path of the "Downloads" folder
2. def get_downloads_folder_path():
3.     if os.name == 'nt':
4.         # Windows system, try to get the "Downloads" folder from environment variables
5.         downloads_folder_path = os.getenv('USERPROFILE')
6.         if downloads_folder_path:
7.             downloads_path = os.path.join(downloads_folder_path, 'Downloads')
8.             if not os.path.exists(downloads_path):
9.                 try:
10.                     os.makedirs(downloads_path)
11.                 except OSError as e:
```

```
                        raise OSError(f"Failed to create Downloads folder. Error: {e}")
                return downloads_path
        else:
            # Non-Windows system, use pathlib to get the "Downloads" folder path
            downloads_path = str(Path.home() / "Downloads")
            if not os.path.exists(downloads_path):
                try:
                    os.makedirs(downloads_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path

```

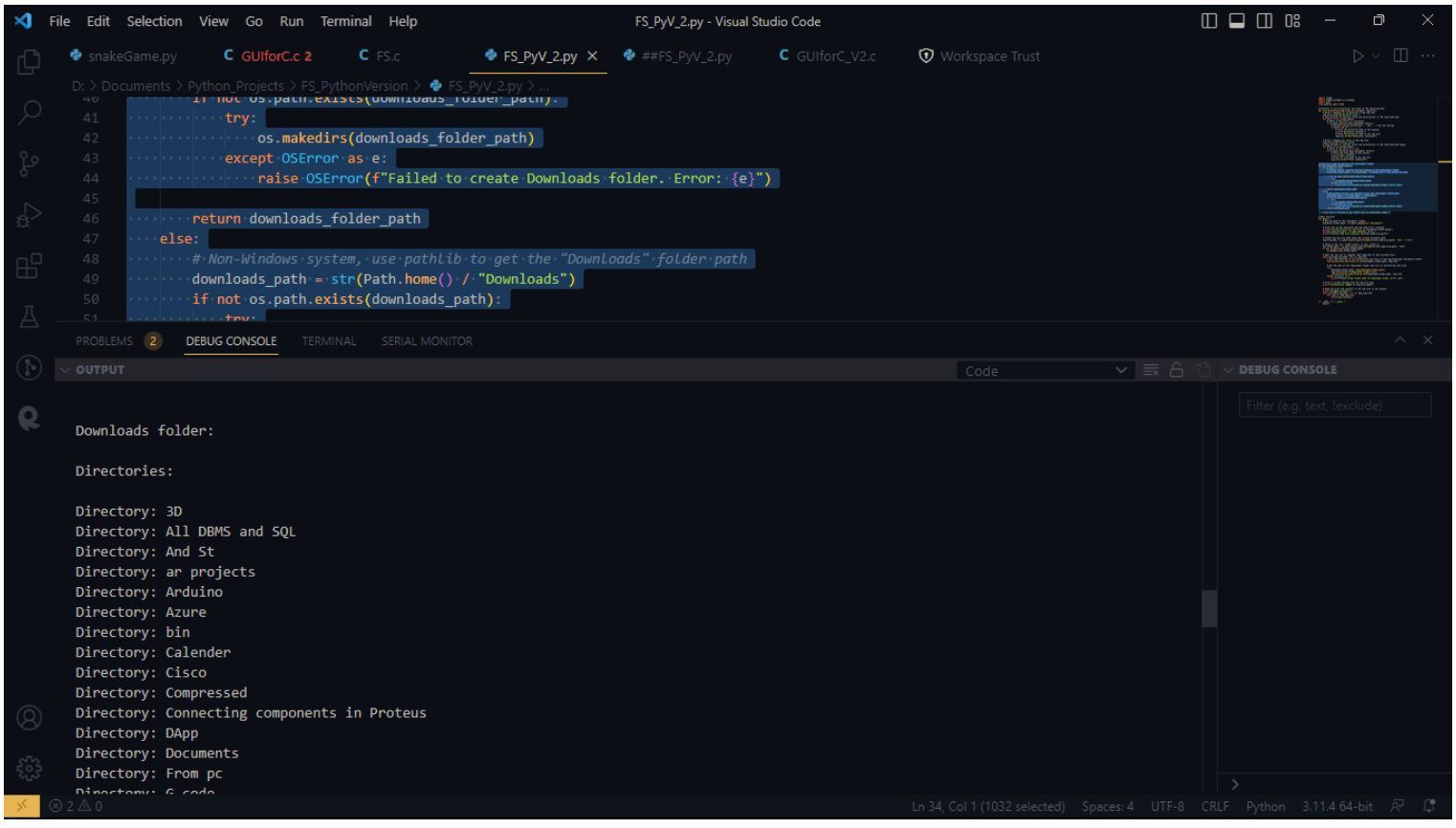
Brought nonn error, but old location



Try 3: Success, semi hardcoded, specifically for my system

The real problem seems to be that the environment variable **USERPROFILE** points to the old location of the "Downloads" folder, which has been moved to another location on your system. Python is still using the old path stored in the environment variable, resulting in the error when trying to access it.
To address this, you can directly specify the new location of the "Downloads" folder in the code instead of relying on the **USERPROFILE** environment variable. Here's an updated version of the function to do that:

```python
# Function to get the path of the "Downloads" folder
def get_downloads_folder_path():
    if os.name == 'nt':
        # Windows system, specify the new location of the "Downloads" folder
        downloads_folder_path = "D:/Downloads"  # Change this to the actual new path

        if not os.path.exists(downloads_folder_path):
            try:
                os.makedirs(downloads_folder_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")

            return downloads_folder_path
        else:
            # Non-Windows system, use pathlib to get the "Downloads" folder path
            downloads_path = str(Path.home() / "Downloads")
            if not os.path.exists(downloads_path):
                try:
                    os.makedirs(downloads_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path

        raise OSError("Failed to get folder path for Downloads folder.")
```



Try 4:

The following works for downloads in my pc, but contents of download folder occur in duplicates:

```
1. import os
2. import ctypes
3. import ctypes.wintypes as wintypes
4. import uuid
5. from pathlib import Path
6. import sys
7.
8. # Function to list directories and files in the specified path
9. def list_directories_and_files_in_root(path, log_file):
10.          # Write a heading for directories in the log file
11.          log_file.write("Directories:\n\n")
12.          # Write a heading for files in the log file
13.          log_file.write("Files:\n\n")
14.
15.          # Loop through all entries (files and directories) in the specified path
16.          for entry in os.listdir(path):
17.              entry_path = os.path.join(path, entry)
18.
19.              # Check if the entry is a directory
20.              if os.path.isdir(entry_path):
21.                  # Exclude special directories '.' and '..' from the listing
22.                  if entry not in ('.', '..'):
23.                      # Print the directory name to the console
24.                      print(f"Directory: {entry}")
25.                      # Write the directory name to the log file
26.                      log_file.write(f"Directory: {entry}\n")
27.
28.              # Check if the entry is a file
29.              elif os.path.isfile(entry_path):
30.                  # Print the file name to the console
31.                  print(f"File: {entry}")
32.                  # Write the file name to the log file
33.                  log_file.write(f"File: {entry}\n")
34.
35.      # Function to get the path of the "Downloads" folder
36.      def get_downloads_folder_path():
37.          if os.name == 'nt':
38.              # Windows system, specify the new location of the "Downloads" folder
39.              downloads_folder_path = "D:/Downloads"  # Change this to the actual new
   path
40.
41.              if not os.path.exists(downloads_folder_path):
42.                  try:
43.                      os.makedirs(downloads_folder_path)
44.                  except OSError as e:
45.                      raise OSError(f"Failed to create Downloads folder. Error: {e}")
46.
47.              return downloads_folder_path
48.          else:
49.              # Non-Windows system, use pathlib to get the "Downloads" folder path
50.              downloads_path = str(Path.home() / "Downloads")
51.              if not os.path.exists(downloads_path):
52.                  try:
53.                      os.makedirs(downloads_path)
54.                  except OSError as e:
```

```
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path.encode('utf-8')

        raise OSError("Failed to get folder path for Downloads folder.")

    # Main function
    def main():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Print the current directory and the code file's location
        print(f"Listing directories and files in {documents_folder_path}")
        current_code_file_path = os.path.abspath(__file__)
        print(f"Current Code File Location: {current_code_file_path}")

        # Create the log file name using the current directory path
        log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin",
"1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(os.path.dirname(current_code_file_path),
"bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Redirect stdout to a temporary file with utf-8 encoding
        temp_stdout = sys.stdout
        with open(log_file_name + ".tmp", "w", encoding="utf-8") as log_file:
            sys.stdout = log_file
            # Call the function to list directories and files in the specified path
(Documents folder)
            list_directories_and_files_in_root(documents_folder_path, log_file)

            # Get the path of the "Downloads" folder and list its directories and
files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path not in (documents_folder_path,
bin_folder_path):
                    print("\n\nDownloads folder:\n\n")
                    list_directories_and_files_in_root(downloads_folder_path,
log_file)
            except Exception as e:
                print(f"\n\nFailed to get folder path for Downloads folder. Error:
{e}")

        # Restore stdout and rename the temporary log file to the final name
        sys.stdout = temp_stdout
        os.replace(log_file_name + ".tmp", log_file_name)

        # Print a success message with the log file name
        print(f"Successfully logged to {log_file_name}")

        # Read and print the contents of the log file to the console
        print("\nLogged Contents:")
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            for line in read_log_file:
```

```
                print(line.rstrip())

    if __name__ == "__main__":
        main()

```

Try 5

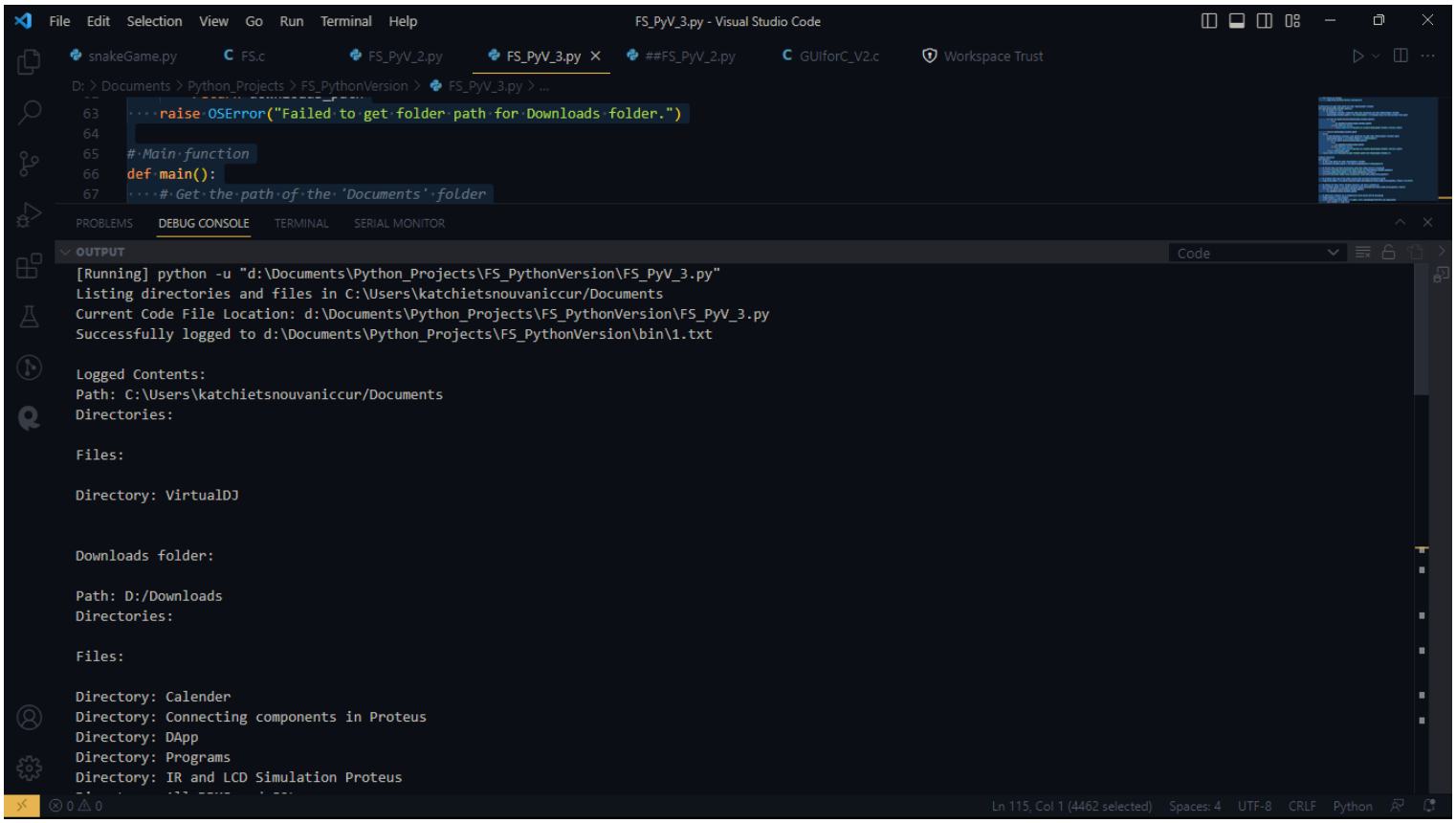
1. This was the most successful one yet:

Everything in D:\Downloads was listed in log file. Semi hard coded path.

```
# Import necessary modules from Python's standard library
import os
import sys
from pathlib import Path

# Function to list directories and files in the specified path
def list_directories_and_files(path, log_file):
    # Write a heading for directories and files in the log file
        log_file.write(f"Path: {path}\n")
        log_file.write("Directories:\n\n")
        log_file.write("Files:\n\n")

        # Initialize two sets to store the names of directories and files
        directories = set()
        files = set()

        # Loop through all entries (files and directories) in the specified path
        for entry in os.listdir(path):
            entry_path = os.path.join(path, entry)

            # Check if the entry is a directory
            if os.path.isdir(entry_path):
                # Exclude special directories '.' and '..' from the listing
                if entry not in ('.', '..'):
                    directories.add(entry)

            # Check if the entry is a file
            elif os.path.isfile(entry_path):
                files.add(entry)

        # Write the directories to the log file
        for directory in directories:
            log_file.write(f"Directory: {directory}\n")

        # Write the files to the log file
        for file in files:
            log_file.write(f"File: {file}\n")

    # Function to get the path of the "Downloads" folder
    def get_downloads_folder_path():
        # Check if the current system is Windows or non-Windows (like macOS, Linux)
        if os.name == 'nt':  # Windows system
            # Specify the new location of the "Downloads" folder for Windows
            downloads_folder_path = "D:/Downloads"  # Change this to your actual new
path on Windows
```

```
46.
47.             # Create the folder if it doesn't exist
48.             if not os.path.exists(downloads_folder_path):
49.                 try:
50.                     os.makedirs(downloads_folder_path)
51.                 except OSError as e:
52.                     raise OSError(f"Failed to create Downloads folder. Error: {e}")
53.
54.             return downloads_folder_path
55.         else:  # Non-Windows system
56.             # Use the pathlib module to get the "Downloads" folder path for non-
    Windows systems
57.             downloads_path = str(Path.home() / "Downloads")
58.
59.             # Create the folder if it doesn't exist
60.             if not os.path.exists(downloads_path):
61.                 try:
62.                     os.makedirs(downloads_path)
63.                 except OSError as e:
64.                     raise OSError(f"Failed to create Downloads folder. Error: {e}")
65.             return downloads_path
66.
67.         # If the code reaches this point, it means the function couldn't determine
    the Downloads folder path
68.         raise OSError("Failed to get folder path for Downloads folder.")
69.
70.     # Main function
71.     def main():
72.         # Get the path of the 'Documents' folder
73.         documents_folder_path = os.path.expanduser("~/Documents")
74.
75.         # Print the current directory and the code file's location
76.         print(f"Listing directories and files in {documents_folder_path}")
77.         current_code_file_path = os.path.abspath(__file__)
78.         print(f"Current Code File Location: {current_code_file_path}")
79.
80.         # Create the log file name using the current directory path
81.         log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin",
    "1.txt")
82.
83.         # Check if the 'bin' folder exists, if not, create it
84.         bin_folder_path = os.path.join(os.path.dirname(current_code_file_path),
    "bin")
85.         if not os.path.exists(bin_folder_path):
86.             os.makedirs(bin_folder_path)
87.
88.         # Redirect stdout to a temporary file with utf-8 encoding
89.         temp_stdout = sys.stdout
90.         with open(log_file_name + ".tmp", "w", encoding="utf-8") as log_file:
91.             sys.stdout = log_file
92.
93.             # Call the function to list directories and files in the "Documents"
    folder
94.             list_directories_and_files(documents_folder_path, log_file)
95.
96.             # Get the path of the "Downloads" folder and list its directories and
    files
```

```
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    log_file.write("\n\nDownloads folder:\n\n")
                    list_directories_and_files(downloads_folder_path, log_file)
            except Exception as e:
                log_file.write(f"\n\nFailed to get folder path for Downloads folder.
  Error: {e}")

        # Restore stdout and rename the temporary log file to the final name
        sys.stdout = temp_stdout
        os.replace(log_file_name + ".tmp", log_file_name)

        # Print a success message with the log file name
        print(f"Successfully logged to {log_file_name}")

        # Read and print the contents of the log file to the console
        print("\nLogged Contents:")
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            for line in read_log_file:
                print(line.rstrip())

    # The program execution starts here
    if __name__ == "__main__":
        main()


```



To correct a certain grouping issue

To correct a certain grouping issue, this syntax could help:

```
1.
2. import os
3. import sys
4. from pathlib import Path
5.
6. # Function to list directories and files in the specified path
7. def list_directories_and_files(path, log_file, title):
8.     # Write a heading for directories and files in the log file
9.     log_file.write(f"{title}:\n")
10.          log_file.write("Directories:\n\n")
11.          log_file.write("Files:\n\n")
12.
13.          # Initialize a set to store the names of directories and files
14.          directories = set()
15.          files = set()
16.
17.          # Loop through all entries (files and directories) in the specified path
18.          for entry in os.listdir(path):
19.              entry_path = os.path.join(path, entry)
20.
21.              # Check if the entry is a directory
22.              if os.path.isdir(entry_path):
23.                  # Exclude special directories '.' and '..' from the listing
24.                  if entry not in ('.', '..'):
25.                      directories.add(entry)
26.
27.              # Check if the entry is a file
28.              elif os.path.isfile(entry_path):
29.                  files.add(entry)
30.
31.          # Write the directories to the log file
32.          for directory in directories:
33.              log_file.write(f"Directory: {directory}\n")
34.
35.          # Write the files to the log file
36.          for file in files:
37.              log_file.write(f"File: {file}\n")
38.
39.      # Rest of the code remains the same...
40.
41.      # Main function
42.      def main():
43.          # Rest of the code remains the same...
44.
45.          # Call the function to list directories and files in the "Documents" folder
46.          list_directories_and_files(documents_folder_path, log_file, "Documents
   folder")
47.
48.          # Get the path of the "Downloads" folder and list its directories and files
49.          try:
50.              downloads_folder_path = get_downloads_folder_path()
51.              if downloads_folder_path != documents_folder_path:
52.                  list_directories_and_files(downloads_folder_path, log_file,
   "Downloads folder")
```

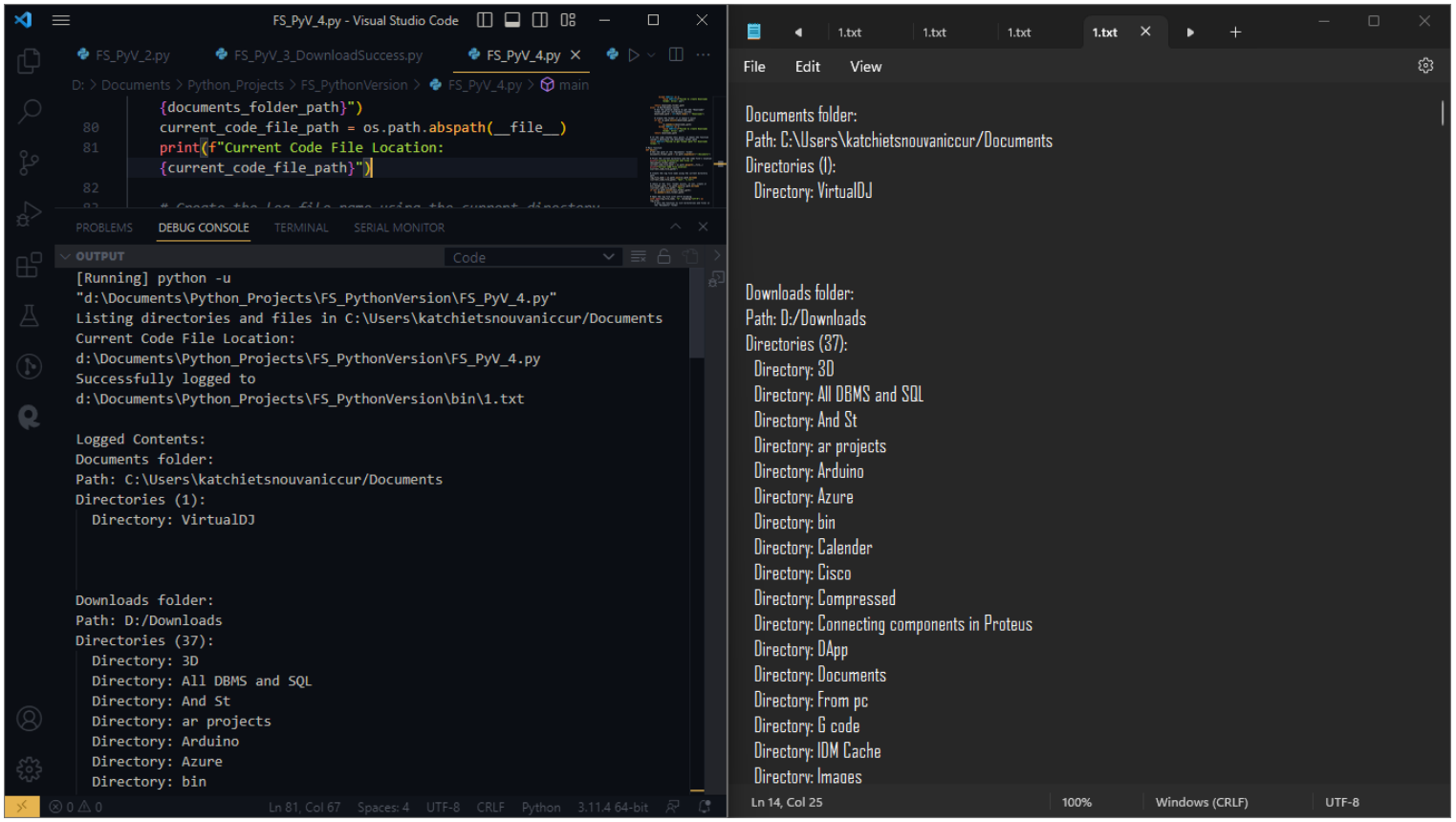
```
        except Exception as e:
            log_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")

        # Rest of the code remains the same...

    if __name__ == "__main__":
        main()

```

Try 6: Best Downloads Folder Logging format

```
1. # Import necessary modules from Python's standard library
2. import os
3. import sys
4. from pathlib import Path
5.
6. # Function to list directories and files in the specified path
7. def list_directories_and_files(path, log_file, title):
8.     # Write a heading for directories and files in the log file
9.     log_file.write(f"{title}:\n")
10.
11.         # Initialize lists to store the names of directories and files
12.         directories = []
13.         files = []
14.
15.         # Loop through all entries (files and directories) in the specified path
16.         for entry in os.listdir(path):
17.             entry_path = os.path.join(path, entry)
18.
19.             # Check if the entry is a directory
20.             if os.path.isdir(entry_path):
21.                 # Exclude special directories '.' and '..' from the listing
22.                 if entry not in ('.', '..'):
23.                     directories.append(entry)
24.
25.             # Check if the entry is a file
26.             elif os.path.isfile(entry_path):
27.                 files.append(entry)
28.
29.         # Write the directories to the log file
30.         if directories:
31.             log_file.write(f"Path: {path}\n")
32.             log_file.write(f"Directories ({len(directories)}):\n")
33.             for directory in directories:
34.                 log_file.write(f"  Directory: {directory}\n")
35.
36.         # Write the files to the log file
37.         if files:
38.             log_file.write(f"Path: {path}\n")
39.             log_file.write(f"Files ({len(files)}):\n")
40.             for file in files:
41.                 log_file.write(f"  File: {file}\n")
42.
43.     # Function to get the path of the "Downloads" folder
44.     def get_downloads_folder_path():
45.         # Check if the current system is Windows or non-Windows (like macOS, Linux)
46.         if os.name == 'nt':  # Windows system
47.             # Specify the new location of the "Downloads" folder for Windows
48.             downloads_folder_path = "D:/Downloads"  # Change this to your actual new
    path on Windows
49.
50.             # Create the folder if it doesn't exist
51.             if not os.path.exists(downloads_folder_path):
52.                 try:
53.                     os.makedirs(downloads_folder_path)
54.                 except OSError as e:
```

```
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")

            return downloads_folder_path
        else:  # Non-Windows system
            # Use the pathlib module to get the "Downloads" folder path for non-Windows systems
            downloads_path = str(Path.home() / "Downloads")

            # Create the folder if it doesn't exist
            if not os.path.exists(downloads_path):
                try:
                    os.makedirs(downloads_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path

        # If the code reaches this point, it means the function couldn't determine the Downloads folder path
        raise OSError("Failed to get folder path for Downloads folder.")

    # Main function
    def main():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Print the current directory and the code file's location
        print(f"Listing directories and files in {documents_folder_path}")
        current_code_file_path = os.path.abspath(__file__)
        print(f"Current Code File Location: {current_code_file_path}")

        # Create the log file name using the current directory path
        log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(os.path.dirname(current_code_file_path), "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents folder")

            # Add separation for major groups (8-line separation)
            log_file.write("\n" * 3)

            # Get the path of the "Downloads" folder and list its directories and files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    list_directories_and_files(downloads_folder_path, log_file, "Downloads folder")
```

```
104.             except Exception as e:
105.                 with open(log_file_name, "a", encoding="utf-8") as log_file:
106.                     log_file.write(f"\n\nFailed to get folder path for Downloads
    folder. Error: {e}")
107.
108.         # Print a success message with the log file name
109.         print(f"Successfully logged to {log_file_name}")
110.
111.         # Read and print the contents of the log file to the console
112.         print("\nLogged Contents:")
113.         with open(log_file_name, "r", encoding="utf-8") as read_log_file:
114.             for line in read_log_file:
115.                 print(line.rstrip())
116.
117.     # The program execution starts here
118.     if __name__ == "__main__":
119.         main()
120.
```



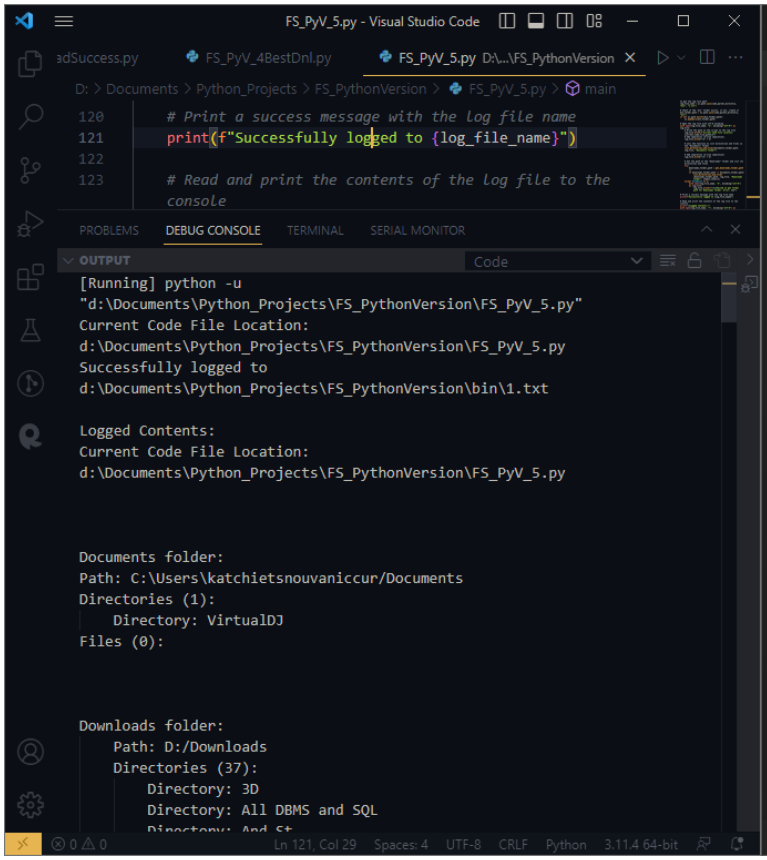
Try 7: Improvement: Root Storage memory, user interface viewing

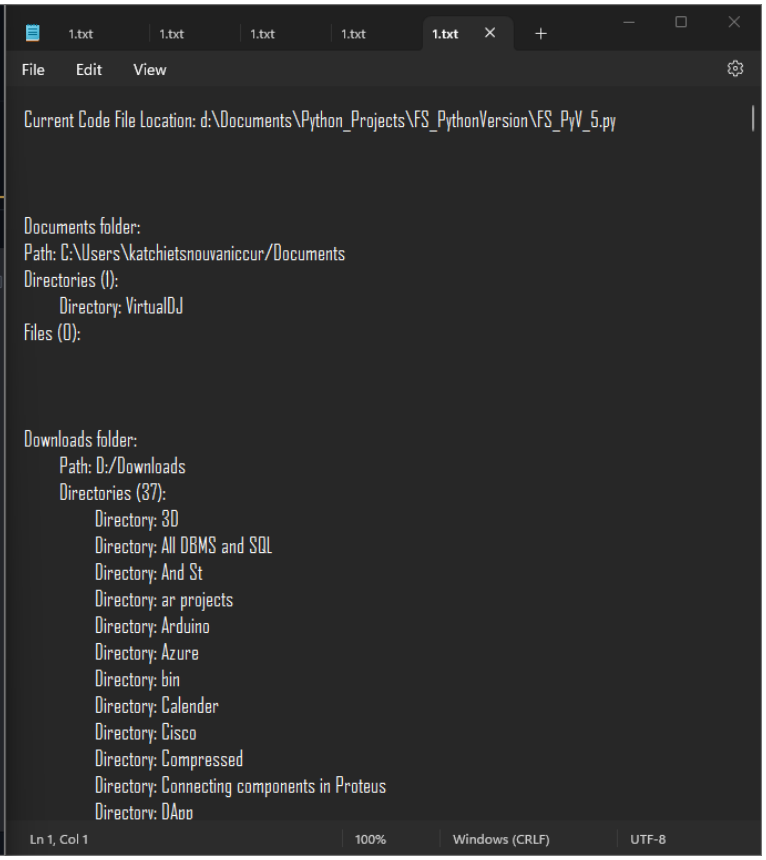
Feature to create the "bin" folder relative to the location of the code file.
Also, the logging had improved formatting

```
# Import necessary modules from Python's standard library
import os
import sys
from pathlib import Path

# Function to list directories and files in the specified path
def list_directories_and_files(path, log_file, title, indent_level=0):
    # Write a heading for directories and files in the log file
    log_file.write(f"{title}:\n")
        indent = "\t" * indent_level
        indent_for_files = "\t" * (indent_level + 1)

        # Initialize lists to store the names of directories and files
        directories = []
        files = []

        # Loop through all entries (files and directories) in the specified path
        for entry in os.listdir(path):
            entry_path = os.path.join(path, entry)

            # Check if the entry is a directory
            if os.path.isdir(entry_path):
                # Exclude special directories '.' and '..' from the listing
                if entry not in ('.', '..'):
                    directories.append(entry)

            # Check if the entry is a file
            elif os.path.isfile(entry_path):
                files.append(entry)

        # Write the directories to the log file
        log_file.write(f"{indent}Path: {path}\n")
        if directories:
            log_file.write(f"{indent}Directories ({len(directories)}):\n")
            for directory in directories:
                log_file.write(f"{indent_for_files}Directory: {directory}\n")
        else:
            log_file.write(f"{indent}Directories (0):\n")

        # Write the files to the log file
        if files:
            log_file.write(f"{indent}Files ({len(files)}):\n")
            for file in files:
                log_file.write(f"{indent_for_files}File: {file}\n")
        else:
            log_file.write(f"{indent}Files (0):\n")

    # Function to get the path of the "Downloads" folder
    def get_downloads_folder_path():
        # Check if the current system is Windows or non-Windows (like macOS, Linux)
        if os.name == 'nt':  # Windows system
            # Specify the new location of the "Downloads" folder for Windows
```

```
            downloads_folder_path = "D:/Downloads"  # Change this to your actual new path on Windows

            # Create the folder if it doesn't exist
            if not os.path.exists(downloads_folder_path):
                try:
                    os.makedirs(downloads_folder_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")

            return downloads_folder_path
        else:  # Non-Windows system
            # Use the pathlib module to get the "Downloads" folder path for non-Windows systems
            downloads_path = str(Path.home() / "Downloads")

            # Create the folder if it doesn't exist
            if not os.path.exists(downloads_path):
                try:
                    os.makedirs(downloads_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path

        # If the code reaches this point, it means the function couldn't determine the Downloads folder path
        raise OSError("Failed to get folder path for Downloads folder.")

    # Main function
    def main():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Print the current directory and the code file's location
        current_code_file_path = os.path.abspath(__file__)
        print(f"Current Code File Location: {current_code_file_path}")

        # Create the log file name using the current directory path
        # Get the parent directory of the code file (where the code is located)
        code_parent_directory = os.path.dirname(current_code_file_path)
        # Combine the parent directory with the 'bin' folder name to get the log file path
        log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(code_parent_directory, "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Write the path of the script to the log file
            log_file.write(f"Current Code File Location: {current_code_file_path}\n")
            # Add separation (3-line separation)
            log_file.write("\n" * 3)

```

```
            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents folder")

            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Get the path of the "Downloads" folder and list its directories and files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    list_directories_and_files(downloads_folder_path, log_file, "Downloads folder", indent_level=1)
            except Exception as e:
                with open(log_file_name, "a", encoding="utf-8") as log_file:
                    log_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")

        # Print a success message with the log file name
        print(f"Successfully logged to {log_file_name}")

        # Read and print the contents of the log file to the console
        print("\nLogged Contents:")
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            for line in read_log_file:
                print(line.rstrip())

    # The program execution starts here
    if __name__ == "__main__":
        main()

```


FS_PyV_5.py - Visual Studio Code
adSuccess.py
FS_PyV_4BestDnl.py
FS_PyV_5.py D:\...\FS_PythonVersion
D: > Documents > Python_Projects > FS_PythonVersion > FS_PyV_5.py > main
120 # Print a success message with the log file name
121 print(f"Successfully logged to {log_file_name}")
122
123 # Read and print the contents of the log file to the console
PROBLEMS
DEBUG CONSOLE
TERMINAL
SERIAL MONITOR
OUTPUT
Code
[Running] python -u "d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_5.py"
Current Code File Location:
d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_5.py
Successfully logged to
d:\Documents\Python_Projects\FS_PythonVersion\bin\1.txt
Logged Contents:
Current Code File Location:
d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_5.py
Documents folder:
Path: C:\Users\katchietsnouvaniccur/Documents
Directories (1):
Directory: VirtualDJ
Files (0):
Downloads folder:
Path: D:/Downloads
Directories (37):
Directory: 3D
Directory: All DBMS and SQL
Ln 121, Col 29
Spaces: 4
UTF-8
CRLF
Python
3.11.4 64-bit


1.txt
File
Edit
View
Current Code File Location: d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_5.py
Documents folder:
Path: C:\Users\katchietsnouvaniccur/Documents
Directories (1):
Directory: VirtualDJ
Files (0):
Downloads folder:
Path: D:/Downloads
Directories (37):
Directory: 3D
Directory: All DBMS and SQL
Directory: And St
Directory: ar projects
Directory: Arduino
Directory: Azure
Directory: bin
Directory: Calender
Directory: Cisco
Directory: Compressed
Directory: Connecting components in Proteus
Directory: DApp
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

File Sniffer + Web Browser for Python:

built-in Flask development server

I ran the following in the terminal:

pip install Flask

Then created a new html under:

D:\Documents\Python_Projects\FS_PythonVersion\templates\ index.html

I then ran the code and navigated to the following link:

http://127.0.0.1:5000/

The html code as follows:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>Directory Listing</title>
5. </head>
6. <body>
7.     <h1>Directory Listing</h1>
8.     {% for line in log_contents %}
9.         {{ line }}<br>
10.    {% endfor %}
11.</body>
12.</html>
13.
```

The python code as follows:

```
1. # Import necessary modules from Python's standard library
2. from concurrent.futures import ThreadPoolExecutor
3. from flask import Flask, render_template
4. import os
5. from pathlib import Path
6. # import sys
7.
8. app = Flask(__name__)
9.
10.# Function to list directories and files in the specified path
11.def list_directories_and_files(path, log_file, title, indent_level=0):
12.    # Write a heading for directories and files in the log file
13.    log_file.write(f"{title}:\n")
14.    indent = "\t" * indent_level
15.    indent_for_files = "\t" * (indent_level + 1)
16.
17.    # Initialize lists to store the names of directories and files
18.    directories = []
19.    files = []
20.
21.    # Loop through all entries (files and directories) in the specified path
22.    for entry in os.listdir(path):
23.        entry_path = os.path.join(path, entry)
24.
25.        # Check if the entry is a directory
26.        if os.path.isdir(entry_path):
27.            # Exclude special directories '.' and '..' from the listing
28.            if entry not in ('.', '..'):
29.                directories.append(entry)
30.
31.        # Check if the entry is a file
```

```
        elif os.path.isfile(entry_path):
            files.append(entry)

    # Write the directories to the log file
    log_file.write(f"{indent}Path: {path}\n")
    if directories:
        log_file.write(f"{indent}Directories ({len(directories)}):\n")
        for directory in directories:
            log_file.write(f"{indent_for_files}Directory: {directory}\n")
    else:
        log_file.write(f"{indent}Directories (0):\n")

    # Write the files to the log file
    if files:
        log_file.write(f"{indent}Files ({len(files)}):\n")
        for file in files:
            log_file.write(f"{indent_for_files}File: {file}\n")
    else:
        log_file.write(f"{indent}Files (0):\n")

# Function to get the path of the "Downloads" folder
def get_downloads_folder_path():
    # Check if the current system is Windows or non-Windows (like macOS, Linux)
    if os.name == 'nt':  # Windows system
        # Specify the new location of the "Downloads" folder for Windows
        downloads_folder_path = "D:/Downloads"  # Change this to your actual new path on Windows

        # Create the folder if it doesn't exist
        if not os.path.exists(downloads_folder_path):
            try:
                os.makedirs(downloads_folder_path)
            except OSError as e:
                raise OSError(f"Failed to create Downloads folder. Error: {e}")

        return downloads_folder_path
    else:  # Non-Windows system
        # Use the pathlib module to get the "Downloads" folder path for non-Windows systems
        downloads_path = str(Path.home() / "Downloads")

        # Create the folder if it doesn't exist
        if not os.path.exists(downloads_path):
            try:
                os.makedirs(downloads_path)
            except OSError as e:
                raise OSError(f"Failed to create Downloads folder. Error: {e}")
        return downloads_path

    # If the code reaches this point, it means the function couldn't determine the Downloads folder path
    raise OSError("Failed to get folder path for Downloads folder.")

# Routes for web application
@app.route('/')
def index():
    # Get the path of the 'Documents' folder
    documents_folder_path = os.path.expanduser("~/Documents")
```

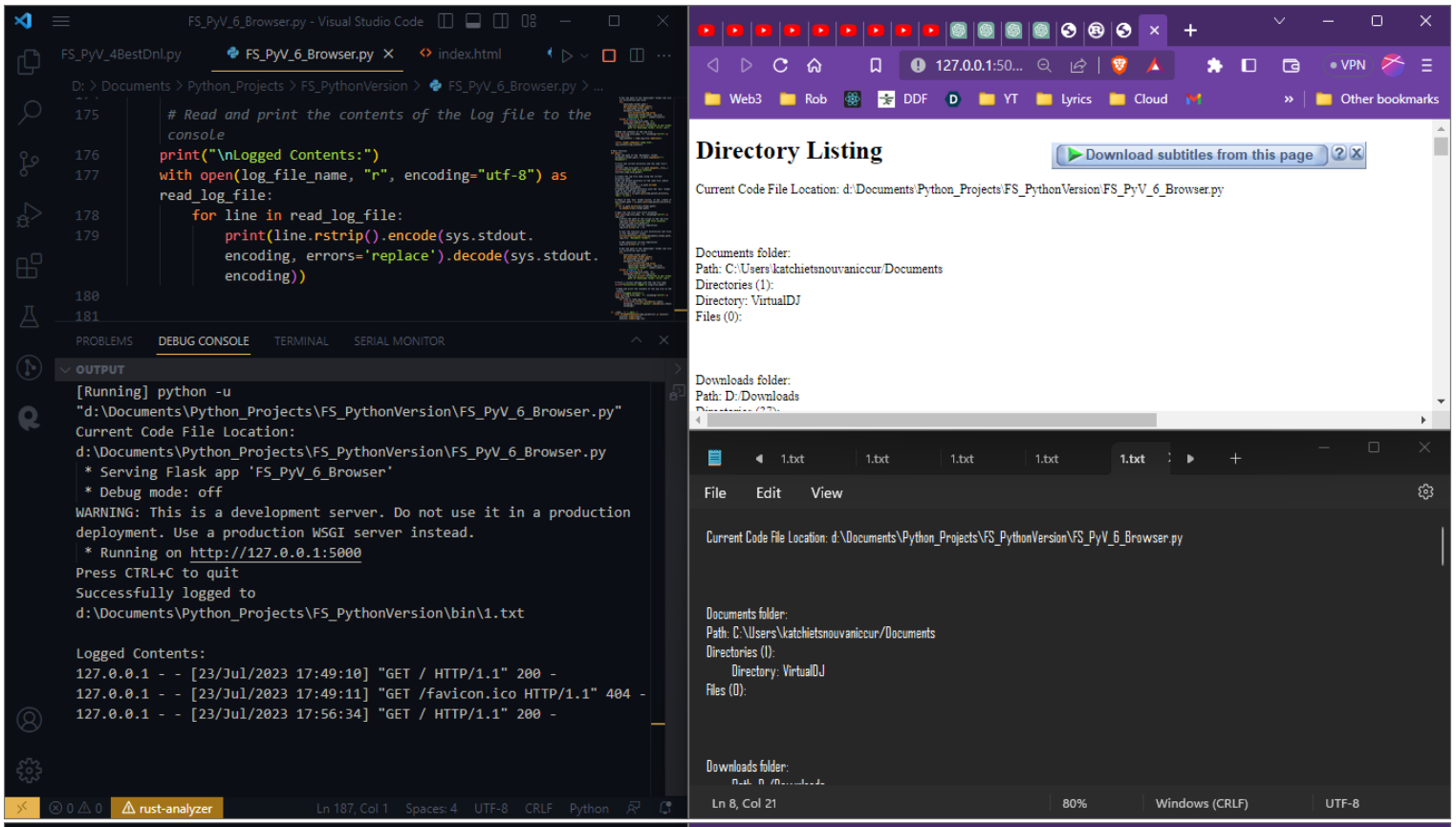
```
87.
88.     # Create the log file name using the current directory path
89.     # Get the parent directory of the code file (where the code is located)
90.     code_parent_directory = os.path.dirname(__file__)
91.     # Combine the parent directory with the 'bin' folder name to get the log file path
92.     log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")
93.
94.     # Check if the 'bin' folder exists, if not, create it
95.     bin_folder_path = os.path.join(code_parent_directory, "bin")
96.     if not os.path.exists(bin_folder_path):
97.         os.makedirs(bin_folder_path)
98.
99.     # Open the log file with utf-8 encoding
100.         with open(log_file_name, "w", encoding="utf-8") as log_file:
101.             # Write the path of the script to the log file
102.             log_file.write(f"Current Code File Location: {__file__}\n")
103.             # Add separation (3-line separation)
104.             log_file.write("\n" * 3)
105.
106.             # Call the function to list directories and files in the "Documents" folder
107.             list_directories_and_files(documents_folder_path, log_file, "Documents
    folder")
108.
109.             # Add separation (3-line separation)
110.             log_file.write("\n" * 3)
111.
112.             # Get the path of the "Downloads" folder and list its directories and files
113.             try:
114.                 downloads_folder_path = get_downloads_folder_path()
115.                 if downloads_folder_path != documents_folder_path:
116.                     list_directories_and_files(downloads_folder_path, log_file,
    "Downloads folder", indent_level=1)
117.             except Exception as e:
118.                 with open(log_file_name, "a", encoding="utf-8") as log_file:
119.                     log_file.write(f"\n\nFailed to get folder path for Downloads folder.
    Error: {e}")
120.
121.         # Read the contents of the log file
122.         with open(log_file_name, "r", encoding="utf-8") as read_log_file:
123.             log_contents = read_log_file.readlines()
124.
125.         return render_template('index.html', log_contents=log_contents)
126.
127.     # Main function
128.     def main():
129.         # Get the path of the 'Documents' folder
130.         documents_folder_path = os.path.expanduser("~/Documents")
131.
132.         # Print the current directory and the code file's location
133.         current_code_file_path = os.path.abspath(__file__)
134.         print(f"Current Code File Location: {current_code_file_path}")
135.
136.         # Create the log file name using the current directory path
137.         # Get the parent directory of the code file (where the code is located)
138.         code_parent_directory = os.path.dirname(current_code_file_path)
139.         # Combine the parent directory with the 'bin' folder name to get the log file
    path
```

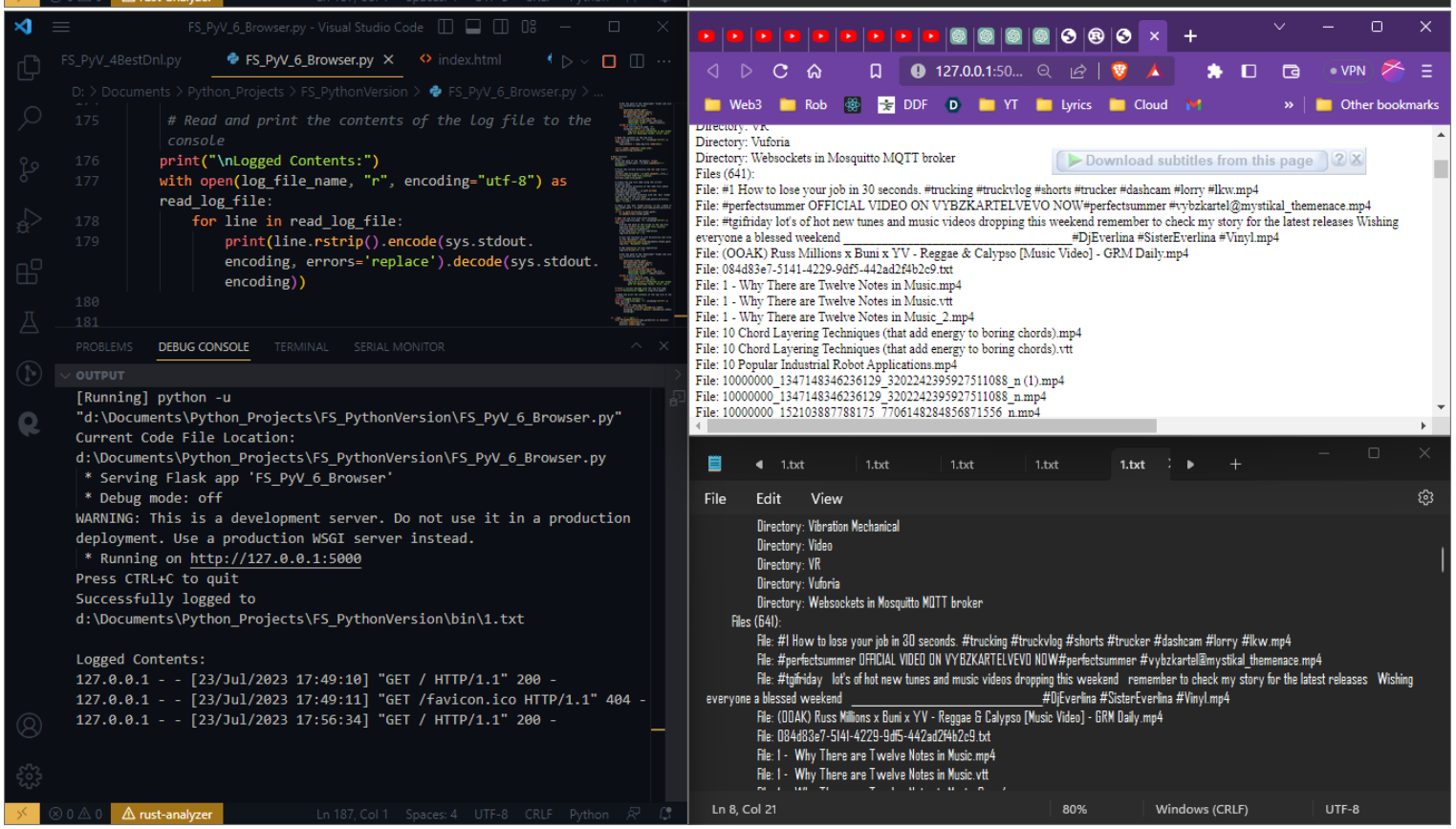
```
        log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(code_parent_directory, "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Write the path of the script to the log file
            log_file.write(f"Current Code File Location: {current_code_file_path}\n")
            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents
   folder")

            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Get the path of the "Downloads" folder and list its directories and files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    list_directories_and_files(downloads_folder_path, log_file,
   "Downloads folder", indent_level=1)
            except Exception as e:
                with open(log_file_name, "a", encoding="utf-8") as log_file:
                    log_file.write(f"\n\nFailed to get folder path for Downloads folder.
   Error: {e}")

        # Print a success message with the log file name
        print(f"Successfully logged to {log_file_name}")

         # Read and print the contents of the log file to the console
        print("\nLogged Contents:")
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            for line in read_log_file:
                print(line.rstrip().encode(sys.stdout.encoding,
   errors='replace').decode(sys.stdout.encoding))

    if __name__ == "__main__":
        with ThreadPoolExecutor(max_workers=2) as executor:
            executor.submit(main)
            executor.submit(app.run)
```

Since you want to execute both the **main()** function and the Flask web server at the same time, you should use the **concurrent.futures** module to run them in parallel. This way, your program can display the output in the console and run the Flask web server simultaneously.
The terminal read:

```
[Running] python -u "d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_6_Browser.py"
Current Code File Location: d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_6_Browser.py
 * Serving Flask app 'FS_PyV_6_Browser'
 * Debug mode: off
```

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production
WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
Successfully logged to d:\Documents\Python_Projects\FS_PythonVersion\bin\1.txt

Logged Contents:
127.0.0.1 - - [23/Jul/2023 17:49:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 17:49:11] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Jul/2023 17:56:34] "GET / HTTP/1.1" 200 -
```





Use a production WSGI server instead.

Gunicorn server

While the built-in Flask development server is convenient for testing and development purposes, it is not recommended for production use. For a production deployment, it is best to use a production-ready WSGI server.
One popular WSGI server for production deployments is Gunicorn (Green Unicorn). Gunicorn can serve your Flask application more efficiently and reliably in a production environment. Here's how you can use Gunicorn with your Flask app:
pip install gunicorn
After installing Gunicorn, you can run your Flask app using Gunicorn from the command line:
gunicorn -w 4 FS_PyV_6_Browser:app
Explanation:

- -w 4: This option specifies the number of worker processes. You can adjust this number based on the resources available on your server. A common practice is to use the number of CPU cores available on your server.
- FS_PyV_6_Browser: This should be the name of your Python script (without the ".py" extension).
- app: This is the name of the Flask app object that Gunicorn should run.

Now, **Gunicorn** will be **running your Flask app**, and it will be accessible at **http://127.0.0.1:8000** (by default).
Please note that for a full production deployment, there are additional considerations to make, such as setting up a **reverse proxy (e.g., Nginx or Apache)** to handle **incoming web requests and serve your application securely**.
Keep in mind that running a production server requires careful configuration and security considerations. If you plan to deploy your application to a public server, it's important to follow best practices and consider using a production-ready web server, such as **Nginx**, to **act as a reverse proxy** in front of **Gunicorn** to **handle incoming web requests securely**. Additionally, you may need to configure your server to run **Gunicorn as a background process** or use **process managers** like **Supervisor** to ensure it runs continuously.

Waitress server (WEB3 Technology)

The ModuleNotFoundError you encountered with Gunicorn indicates that the fcntl module is missing. The fcntl module is a Unix-specific module and is not available on Windows, which is causing this issue.
Since Gunicorn is primarily designed for Unix-based systems and is not well-suited for Windows, you can use a different WSGI server that is Windows-compatible. One such option is to use Waitress, a pure-Python WSGI server that works on both Unix and Windows systems.
Here's how you can install Waitress and run your Flask app using it:

1. Install Waitress using pip:

pip install waitress
Run your Flask app using Waitress:
waitress-serve --call FS_PyV_6_Browser:app
Explanation:

- --call: This option tells Waitress to load the WSGI app using a callable object.

Now, your Flask app should be running using Waitress on Windows, and it will be accessible at http://127.0.0.1:8080 (by default).
Please note that Waitress is a production-ready server and is well-suited for Windows deployments. However, for production deployments on Unix-based systems, you should still consider using Gunicorn or other production-grade WSGI servers like uWSGI. Remember that additional configurations may be necessary for a full production deployment, such as using a reverse proxy for handling incoming requests and ensuring security.
Terminal:

```
Successfully installed waitress-2.1.2
PS D:\Documents\Python_Projects\FS_PythonVersion> waitress-serve --call FS_PyV_6_Browser:app
Traceback (most recent call last):
  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "C:\Users\katchietsnouvaniccur\AppData\Local\Programs\Python\Python311\Scripts\waitress-serve.exe\__main__.py",
line 7, in <module>
  File "C:\Users\katchietsnouvaniccur\AppData\Local\Programs\Python\Python311\Lib\site-packages\waitress\runner.py",
line 293, in run
    app = app()
          ^^^^^
TypeError: Flask.__call__() missing 2 required positional arguments: 'environ' and 'start_response'
```

Solution:
The error you encountered with Waitress indicates that the Flask app object is not being passed correctly. Waitress expects a callable object, and it seems that the Flask app is not being passed as expected.
To fix this, you can modify the way you run the Waitress server as follows:
waitress-serve --call "FS_PyV_6_Browser:create_app"
In your Flask code, update the app variable declaration at the top to create_app() as shown below:

```
1. from flask import Flask, render_template
2. import os
3. from pathlib import Path
4.
5. # Your functions...
6.
7. # Create the Flask app using a function
8. def create_app():
9.     app = Flask(__name__)
10.
11.    # Your routes and other app configuration here...
12.    # Routes for web application
13.    @app.route('/')
14.    def index():
15.        # ... your existing code ...
16.
17.    return app
18.
19.# Main function (if needed)
20.def main():
21.    # ... your existing code ...
22.
23.# The program execution starts here
24.if __name__ == "__main__":
25.    with ThreadPoolExecutor(max_workers=2) as executor:
26.        executor.submit(main)
27.        executor.submit(create_app().run)
28.
29.
```

With these changes, you should be able to run the Waitress server correctly. The create_app() function will be called by Waitress to create the Flask app, and the server should start without any errors.
Now, your Flask app should be running using Waitress on Windows, and it will be accessible at **http://127.0.0.1:8080** (by default).
If you encounter any further issues, please let me know, and I'll be glad to assist you!

Errors:
It seems that there might be an issue with the setup or configuration of the create_app() function in your code.
Based on your current code, it looks like the create_app() function might not be present in your Flask application script (FS_PyV_6_Browser.py or FS_PyV_6_BrowserSecure.py).
Here's how you can modify your Flask application script to include the create_app() function:

1. Open your Flask application script (e.g., FS_PyV_6_Browser.py).
2. Add the create_app() function before the app variable declaration. It should look like this:

```
1. from flask import Flask, render_template
2. import os
3. from pathlib import Path
```

```
4.
5. # Your functions...
6.
7. # Create the Flask app using a function
8. def create_app():
9.     app = Flask(__name__)
10.
11.    # Your routes and other app configuration here...
12.    # Routes for web application
13.    @app.route('/')
14.    def index():
15.        # ... your existing code ...
16.
17.    return app
18.
19.# Your other functions and main function...
20.
21.# The program execution starts here
22.if __name__ == "__main__":
23.    main()
24.
```

3. Save the changes to the script.
4. Now, try running the Waitress server again using the following command:

waitress-serve --call "FS_PyV_6_Browser:create_app"

Ensure that you are running this command from the same directory where your Flask application script (FS_PyV_6_Browser.py) is located.
If you still encounter issues, please double-check the create_app() function in your Flask application script and make sure it is defined correctly. Additionally, ensure that the script contains the necessary imports and that there are no typos or syntax errors in the code.

Terminal:
PS D:\Documents\Python_Projects\FS_PythonVersion> waitress-serve --call "FS_PyV_6_Browser:create_app"
INFO:waitress:Serving on http://0.0.0.0:8080
Great! The Waitress server is now running and serving your Flask app. It's listening on http://0.0.0.0:8080, which means it's accessible from any IP address on the local network.
To access the web application, open a web browser and navigate to http://localhost:8080 or http://127.0.0.1:8080. You should see your Flask app's home page with the contents listed in the log file.
If you want to access the app from another device on the same network, you can use the IP address of the machine where the server is running instead of "localhost" or "127.0.0.1". For example, if the IP address of the machine is 192.168.1.100, you can access the app from another device using http://192.168.1.100:8080.
so i can access it even on another computer?
Yes, you can access the Flask app served by Waitress on another computer as long as both computers are on the same local network. Here's how you can do it:

1. Find the IP address of the computer where the Waitress server is running: On the computer running the server, open a command prompt or terminal and type the following command:

ipconfig # For Windows
or

ifconfig # For macOS and Linux

1. Look for the network adapter that is connected to your local network (usually under "Ethernet adapter" or "Wi-Fi adapter" on Windows). Note down the IPv4 address of the adapter; it will look something like 192.168.1.xxx or 10.0.0.xxx.
2. Make sure both computers are connected to the same local network: To access the server from another computer, both computers need to be connected to the same Wi-Fi network or be part of the same local network.

3. Access the Flask app from another computer: On the other computer, open a web browser and enter the IP address and port number of the computer running the server. For example, if the server's IP address is 192.168.1.100 and it's running on port 8080, you can access the app by entering the following URL in the web browser:

http://192.168.1.100:8080

1. The Flask app should now be accessible on the other computer, and you can interact with it just like you would on the local machine.

Keep in mind that when accessing the app from another computer on the local network, the firewall settings on the server computer may need to allow incoming connections to the port used by Waitress (8080 in this case). If you encounter any connection issues, check the firewall settings and ensure that the port is open for incoming connections on the server computer.

Errors:

If the terminal appears to be stuck and the webpage is not accessible, it indicates that the Waitress server might not be running correctly or there could be an issue with the network configuration.
Here are a few troubleshooting steps you can try:

1. Check Firewall Settings: Make sure the firewall on the server computer allows incoming connections to the port (8080) used by Waitress. Firewalls can sometimes block external connections, leading to connectivity issues.
2. Restart the Waitress Server: Press Ctrl + C in the terminal where Waitress is running to stop the server. Then, re-run the waitress-serve command:

waitress-serve --call "FS_PyV_6_Browser:create_app"

Check if the server starts up correctly and displays the "Serving on http://0.0.0.0:8080" message again.

3. Verify Network Connectivity: Ensure that both the server computer and the computer from which you are trying to access the Flask app are connected to the same local network. Verify that you can ping the server computer from the other computer and vice versa using the IP address obtained from the ipconfig or ifconfig command.
4. Use the Server's Local IP Address: Instead of using 0.0.0.0 in the URL, try using the local IP address of the server computer. Replace 0.0.0.0 in the URL with the IP address obtained from the ipconfig or ifconfig command. For example, if the server's IP address is 192.168.1.100, access the app using:

http://192.168.1.100:8080

5. Check for Port Conflicts: Ensure that no other application is using port 8080 on the server computer. If another application is using the same port, it can prevent Waitress from binding to that port and serving the app.
6. Restart the Server Computer: Sometimes, network configurations may require a computer restart to take effect properly. If possible, try restarting the server computer and re-run the Waitress server.

If you have tried the above steps and still face issues, there might be other configuration or network-related problems that need further investigation. In that case, consider seeking help from a network administrator or IT professional to assist in troubleshooting the connectivity problem.

Web3 Era Code Try 1

The following code wuld have been my 1st real WEB3 code had it worked but somehow it hanged:

1. S

```
# Import necessary modules from Python's standard library
# from concurrent.futures import ThreadPoolExecutor
from flask import Flask, render_template
import os
from pathlib import Path
import sys

```

```python
# Create the Flask app using a function
def create_app():
    app = Flask(__name__)
    # Your routes and other app configuration here...
    # Routes for web application
    @app.route('/')
    def index():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Create the log file name using the current directory path
        # Get the parent directory of the code file (where the code is located)
        code_parent_directory = os.path.dirname(__file__)
        # Combine the parent directory with the 'bin' folder name to get the log file path
        log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(code_parent_directory, "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Write the path of the script to the log file
            log_file.write(f"Current Code File Location: {__file__}\n")
            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents folder")

            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Get the path of the "Downloads" folder and list its directories and files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    list_directories_and_files(downloads_folder_path, log_file, "Downloads folder", indent_level=1)
            except Exception as e:
                with open(log_file_name, "a", encoding="utf-8") as log_file:
                    log_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")

        # Read the contents of the log file
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            log_contents = read_log_file.readlines()

        return render_template('index.html', log_contents=log_contents)
    return app


# Function to list directories and files in the specified path
def list_directories_and_files(path, log_file, title, indent_level=0):
    # Write a heading for directories and files in the log file
    log_file.write(f"{title}:\n")
```

```
    indent = "\t" * indent_level
    indent_for_files = "\t" * (indent_level + 1)

    # Initialize lists to store the names of directories and files
    directories = []
    files = []

    # Loop through all entries (files and directories) in the specified path
    for entry in os.listdir(path):
        entry_path = os.path.join(path, entry)

        # Check if the entry is a directory
        if os.path.isdir(entry_path):
            # Exclude special directories '.' and '..' from the listing
            if entry not in ('.', '..'):
                directories.append(entry)

        # Check if the entry is a file
        elif os.path.isfile(entry_path):
            files.append(entry)

    # Write the directories to the log file
    log_file.write(f"{indent}Path: {path}\n")
    if directories:
        log_file.write(f"{indent}Directories ({len(directories)}):\n")
        for directory in directories:
            log_file.write(f"{indent_for_files}Directory: {directory}\n")
    else:
        log_file.write(f"{indent}Directories (0):\n")

    # Write the files to the log file
    if files:
        log_file.write(f"{indent}Files ({len(files)}):\n")
        for file in files:
            log_file.write(f"{indent_for_files}File: {file}\n")
    else:
        log_file.write(f"{indent}Files (0):\n")

  # Function to get the path of the "Downloads" folder
  def get_downloads_folder_path():
      # Check if the current system is Windows or non-Windows (like macOS, Linux)
      if os.name == 'nt':  # Windows system
          # Specify the new location of the "Downloads" folder for Windows
          downloads_folder_path = "D:/Downloads"  # Change this to your actual new path
on Windows

          # Create the folder if it doesn't exist
          if not os.path.exists(downloads_folder_path):
              try:
                  os.makedirs(downloads_folder_path)
              except OSError as e:
                  raise OSError(f"Failed to create Downloads folder. Error: {e}")

          return downloads_folder_path
      else:  # Non-Windows system
          # Use the pathlib module to get the "Downloads" folder path for non-Windows
systems
```

```
            downloads_path = str(Path.home() / "Downloads")

            # Create the folder if it doesn't exist
            if not os.path.exists(downloads_path):
                try:
                    os.makedirs(downloads_path)
                except OSError as e:
                    raise OSError(f"Failed to create Downloads folder. Error: {e}")
            return downloads_path

        # If the code reaches this point, it means the function couldn't determine the Downloads folder path
        raise OSError("Failed to get folder path for Downloads folder.")



    # Main function
    def main():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Print the current directory and the code file's location
        current_code_file_path = os.path.abspath(__file__)
        print(f"Current Code File Location: {current_code_file_path}")

        # Create the log file name using the current directory path
        # Get the parent directory of the code file (where the code is located)
        code_parent_directory = os.path.dirname(current_code_file_path)
        # Combine the parent directory with the 'bin' folder name to get the log file path
        log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(code_parent_directory, "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Write the path of the script to the log file
            log_file.write(f"Current Code File Location: {current_code_file_path}\n")
            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents folder")

            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Get the path of the "Downloads" folder and list its directories and files
            try:
                downloads_folder_path = get_downloads_folder_path()
                if downloads_folder_path != documents_folder_path:
                    list_directories_and_files(downloads_folder_path, log_file, "Downloads folder", indent_level=1)
```

```
            except Exception as e:
                with open(log_file_name, "a", encoding="utf-8") as log_file:
                    log_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")

        # Print a success message with the log file name
        print(f"Successfully logged to {log_file_name}")

         # Read and print the contents of the log file to the console
        print("\nLogged Contents:")
        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
            for line in read_log_file:
                print(line.rstrip().encode(sys.stdout.encoding, errors='replace').decode(sys.stdout.encoding))

    # The program execution starts here
    if __name__ == "__main__":
        main()
```

Summary of the WEB3 Code:

It defines a Flask app using the **create_app()** function and sets up a route for the homepage (**/**). When a user accesses the homepage, it will list the directories and files in the "Documents" folder and the "Downloads" folder (if available) and log the information to a text file named **1.txt** in the **bin** folder.

The **main()** function prints the current directory and code file's location, creates the log file, lists the directories and files in the "Documents" folder, lists the directories and files in the "Downloads" folder (if available), and prints the logged contents to the console.

To summarize, this code should create a Flask web application that displays a list of directories and files in the "Documents" and "Downloads" folders when accessed through the homepage. Additionally, it logs this information to a text file and prints the same information to the console.

If you encounter any issues running the code or accessing the web application, please check the previous troubleshooting steps and ensure that the Waitress server is running as expected. If you have any further questions or run into any problems, feel free to ask for assistance.

To check if the Waitress server is running, you can open a web browser and try to access the URL where the server is supposed to be running. In your case, the server is set to run on http://0.0.0.0:8080/.

Here's how you can check if the Waitress server is running:

1. Open your web browser.
2. Enter the URL http://0.0.0.0:8080/ in the address bar.
3. Press Enter or click "Go."

If the Waitress server is running and everything is set up correctly, you should see the web application's homepage displaying the list of directories and files in the "Documents" and "Downloads" folders.

If the server is running and accessible, but you are unable to access it from your computer, it could be a firewall or network-related issue. Make sure that the port 8080 is not blocked by any firewall or security software and that your computer can communicate with the server.

If you still encounter issues, you can also try accessing the server using the IP address of the computer where the server is running. For example, if the IP address of the computer running the server is 192.168.1.100, you can try accessing http://192.168.1.100:8080/ in the web browser.

Additionally, you can also check the terminal or command prompt where you started the Waitress server. If the server is running, it should show logs indicating that it is serving requests.

If you encounter any errors or issues while trying to access the server or if the server is not running, please double-check the previous steps to ensure everything is set up correctly. If you need further assistance, feel free to ask.

You need to remove the portion of the **main()** function that is not related to serving the Flask app with Waitress, as it causes your script to run some unrelated code after the server starts. Keep the **main()** function focused on serving the app, and you don't need to call the function again later in the script

Once you have saved the code in a Python script (for example, **app.py**), you can run the script using the **python** command in your terminal or command prompt. Here's the step-by-step process:

1. Save the code in a file named **app.py** (or any other preferred name).
2. Open your terminal or command prompt.
3. Navigate to the directory where you saved the **app.py** file using the **cd** command. For example:

cd /path/to/your/directory

4. Once you are in the correct directory, run the script with the following command:

python app.py

5. The Waitress server will start running, and you should see a message like:

INFO:waitress:Serving on http://0.0.0.0:8080

6. Now, you can access the web page at http://0.0.0.0:8080/ from your web browser. If you are accessing the web page from another device on the same network, replace **0.0.0.0** with the IP address of the computer running the server.

```python
# Import necessary modules from Python's standard library
from flask import Flask, render_template
import os
from pathlib import Path
from waitress import serve

# Create the Flask app using a function
def create_app():
    app = Flask(__name__)
    # Your routes and other app configuration here...
    # Routes for web application
    @app.route('/')
    def index():
        # Get the path of the 'Documents' folder
        documents_folder_path = os.path.expanduser("~/Documents")

        # Create the log file name using the current directory path
        # Get the parent directory of the code file (where the code is located)
        code_parent_directory = os.path.dirname(__file__)
        # Combine the parent directory with the 'bin' folder name to get the log file path
        log_file_name = os.path.join(code_parent_directory, "bin", "1.txt")

        # Check if the 'bin' folder exists, if not, create it
        bin_folder_path = os.path.join(code_parent_directory, "bin")
        if not os.path.exists(bin_folder_path):
            os.makedirs(bin_folder_path)

        # Open the log file with utf-8 encoding
        with open(log_file_name, "w", encoding="utf-8") as log_file:
            # Write the path of the script to the log file
            log_file.write(f"Current Code File Location: {__file__}\n")
            # Add separation (3-line separation)
            log_file.write("\n" * 3)

            # Call the function to list directories and files in the "Documents" folder
            list_directories_and_files(documents_folder_path, log_file, "Documents folder")

            # Add separation (3-line separation)
            log_file.write("\n" * 3)
```

```
40.
41.            # Get the path of the "Downloads" folder and list its directories and files
42.            try:
43.                downloads_folder_path = get_downloads_folder_path()
44.                if downloads_folder_path != documents_folder_path:
45.                    list_directories_and_files(downloads_folder_path, log_file, "Downloads
   folder", indent_level=1)
46.            except Exception as e:
47.                with open(log_file_name, "a", encoding="utf-8") as log_file:
48.                    log_file.write(f"\n\nFailed to get folder path for Downloads folder.
   Error: {e}")
49.
50.        # Read the contents of the log file
51.        with open(log_file_name, "r", encoding="utf-8") as read_log_file:
52.            log_contents = read_log_file.readlines()
53.
54.        return render_template('index.html', log_contents=log_contents)
55.    return app
56.
57.# Function to list directories and files in the specified path
58.def list_directories_and_files(path, log_file, title, indent_level=0):
59.    # Write a heading for directories and files in the log file
60.    log_file.write(f"{title}:\n")
61.    indent = "\t" * indent_level
62.    indent_for_files = "\t" * (indent_level + 1)
63.
64.    # Initialize lists to store the names of directories and files
65.    directories = []
66.    files = []
67.
68.    # Loop through all entries (files and directories) in the specified path
69.    for entry in os.listdir(path):
70.        entry_path = os.path.join(path, entry)
71.
72.        # Check if the entry is a directory
73.        if os.path.isdir(entry_path):
74.            # Exclude special directories '.' and '..' from the listing
75.            if entry not in ('.', '..'):
76.                directories.append(entry)
77.
78.        # Check if the entry is a file
79.        elif os.path.isfile(entry_path):
80.            files.append(entry)
81.
82.    # Write the directories to the log file
83.    log_file.write(f"{indent}Path: {path}\n")
84.    if directories:
85.        log_file.write(f"{indent}Directories ({len(directories)}):\n")
86.        for directory in directories:
87.            log_file.write(f"{indent_for_files}Directory: {directory}\n")
88.    else:
89.        log_file.write(f"{indent}Directories (0):\n")
90.
91.    # Write the files to the log file
92.    if files:
93.        log_file.write(f"{indent}Files ({len(files)}):\n")
94.        for file in files:
```

```
            log_file.write(f"{indent_for_files}File: {file}\n")
    else:
        log_file.write(f"{indent}Files (0):\n")

# Function to get the path of the "Downloads" folder
def get_downloads_folder_path():
    # Check if the current system is Windows or non-Windows (like macOS, Linux)
    if os.name == 'nt':  # Windows system
        # Specify the new location of the "Downloads" folder for Windows
        downloads_folder_path = "D:/Downloads"  # Change this to your actual new path on Windows

        # Create the folder if it doesn't exist
        if not os.path.exists(downloads_folder_path):
            try:
                os.makedirs(downloads_folder_path)
            except OSError as e:
                raise OSError(f"Failed to create Downloads folder. Error: {e}")

        return downloads_folder_path
    else:  # Non-Windows system
        # Use the pathlib module to get the "Downloads" folder path for non-Windows systems
        downloads_path = str(Path.home() / "Downloads")

        # Create the folder if it doesn't exist
        if not os.path.exists(downloads_path):
            try:
                os.makedirs(downloads_path)
            except OSError as e:
                raise OSError(f"Failed to create Downloads folder. Error: {e}")
        return downloads_path

    # If the code reaches this point, it means the function couldn't determine the Downloads folder path
    raise OSError("Failed to get folder path for Downloads folder.")

# Main function
def main():
    # Create the Flask app using the 'create_app()' function
    app = create_app()

    # Specify the host and port for Waitress to listen on
    host = '0.0.0.0'  # Listen on all available network interfaces
    port = 8080

    # Use Waitress to serve the app
    serve(app, host=host, port=port)

# The program execution starts here
if __name__ == "__main__":
    main()

```

File paths

NB look at difference btn

```
# Get the path of the 'Documents' folder
    documents_folder_path = os.path.expanduser("~/Documents")
```

and

```
# Get the path of the 'Downloads' folder
    downloads_folder_path = get_downloads_folder_path()
```

and

```
# Create the log file name using the current directory path
log_file_name = os.path.join(downloads_folder_path, "1.txt")
```

```
# Create the log file name using the downloads folder path
log_file_name = os.path.join(os.path.dirname(current_code_file_path), "bin", "1.txt")
```

List of Errors:

File: EXPOSING the Music Industry One Day at a Time.mp4 File: EXPOSING the Music Industry One Day at a Time.vtt Traceback (most recent call last): File "d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_6_Browser.py", line 134, in <module> main() File "d:\Documents\Python_Projects\FS_PythonVersion\FS_PyV_6_Browser.py", line 129, in main print(line.rstrip()) File "C:\Users\katchietsnouvaniccur\AppData\Local\Programs\Python\Python311\Lib\encodings\cp1252.py", line 19, in encode return codecs.charmap_encode(input,self.errors,encoding_table)[0] ^^ UnicodeEncodeError: 'charmap' codec can't encode character '\u03c0' in position 12: character maps to <undefined> [Done] exited with code=1 in 3.003 seconds

The issue you are facing is related to encoding when printing the log contents to the console. The error message suggests that there is a UnicodeEncodeError when trying to encode a character that cannot be handled by the 'charmap' codec. To resolve this issue, you can change the encoding used when printing to the console.

From:

```
# Read and print the contents of the log file to the console
    print("\nLogged Contents:")
    with open(log_file_name, "r", encoding="utf-8") as read_log_file:
        for line in read_log_file:
            print(line.rstrip())
```

To:

```
# Read and print the contents of the log file to the console
    print("\nLogged Contents:")
    with open(log_file_name, "r", encoding="utf-8") as read_log_file:
        for line in read_log_file:
            print(line.rstrip().encode(sys.stdout.encoding, errors='replace').decode(sys.stdout.encoding))
```