PYTHON  
Learning Pygame

Table of Contents

[PYTHON Learning Pygame 1](#_Toc141039781)

[Preliminaries 2](#_Toc141039782)

[Building a Snake Game 3](#_Toc141039783)

[Music Player 5](#_Toc141039784)

[Trial Error for music app to open a directory and play music 7](#_Toc141039785)

[1st Working music App 9](#_Toc141039786)

[Generating .exe file 11](#_Toc141039787)

[Converting File Snffer.c file to FileSNiffer.py file 12](#_Toc141039788)

[Adding multiple default folder 14](#_Toc141039789)

[Trial And Error Adding Downloads Folder 15](#_Toc141039790)

[Accessing Downloads folder using new method 15](#_Toc141039791)

[Accessing multiple operating system Downloads folders 19](#_Toc141039792)

[Getting user’s profile name and adding path to the gotten path 22](#_Toc141039793)

[Try 1 22](#_Toc141039794)

[Try 2 22](#_Toc141039795)

[Try 3: Success, semi hardcoded, specifically for my system 24](#_Toc141039796)

[Try 4: 26](#_Toc141039797)

[Try 5 28](#_Toc141039798)

[Try 6: Best Downloads Folder Logging format 34](#_Toc141039799)

[Try 7: Improvement: Root Storage memory, user interface viewing 37](#_Toc141039800)

[File Sniffer + Web Browser for Python: 41](#_Toc141039801)

[Use a production WSGI server instead. 47](#_Toc141039802)

[Gunicorn server 47](#_Toc141039803)

[Waitress server (WEB3 Technology) 47](#_Toc141039804)

[Web3 Era Code Try 1 50](#_Toc141039805)

[Summary of the WEB3 Code: 54](#_Toc141039806)

[File paths 59](#_Toc141039807)

### 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:

sqlCopy 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.

Building a Snake Game  
The following code was able to run for pygame

1. *# Import the Pygame library, which allows us to create graphical applications and games.*
2. import pygame
3. *# Initialize Pygame. This prepares Pygame for use.*
4. pygame.init()
5. *# Define some colors using RGB values.*
6. white = (255, 255, 255)  *# For the background and the color of the snake's head.*
7. black = (0, 0, 0)        *# For the outline of the snake's head.*
8. red = (255, 0, 0)        *# Not used in this code, but can be used for other elements.*
9. *# Create a window (display surface) with a resolution of 800x600 pixels, where the game will be drawn.*
10. dis = pygame.display.set\_mode((800, 600))
11. *# Set the title of the window to "Snake Game".*
12. pygame.display.set\_caption('Snake Game')
13. *# Initialize a variable to control the main game loop.*
14. game\_over = False
15. *# Initialize the coordinates of the snake's head (square).*
16. x1 = 300
17. y1 = 300
18. *# Initialize variables to control the change in position of the snake's head (movement).*
19. x1\_change = 0
20. y1\_change = 0
21. *# Create a clock object to control the frame rate of the game.*
22. clock = pygame.time.Clock()
23. *# The main game loop.*
24. while not game\_over:
25. *# Handle events such as user input (e.g., pressing the close button on the window or arrow keys).*
26. for event in pygame.event.get():
27. if event.type == pygame.QUIT:
28. game\_over = True
29. if event.type == pygame.KEYDOWN:
30. *# Check if the arrow keys are pressed and update the movement variables accordingly.*
31. if event.key == pygame.K\_LEFT:
32. x1\_change = -10
33. y1\_change = 0
34. elif event.key == pygame.K\_RIGHT:
35. x1\_change = 10
36. y1\_change = 0
37. elif event.key == pygame.K\_UP:
38. y1\_change = -10
39. x1\_change = 0
40. elif event.key == pygame.K\_DOWN:
41. y1\_change = 10
42. x1\_change = 0
43. *# Update the position of the snake's head based on the movement variables.*
44. x1 += x1\_change
45. y1 += y1\_change
46. *# Clear the screen by filling it with a white color.*
47. dis.fill(white)
48. *# Draw a black square representing the snake's head at the updated position.*
49. pygame.draw.rect(dis, black, [x1, y1, 10, 10])
50. *# Update the display to show the changes made in the current iteration of the loop.*
51. pygame.display.update()
52. *# Control the frame rate of the game (30 frames per second).*
53. clock.tick(30)
54. *# After the game loop ends (when game\_over becomes True):*
55. *# Uninitialize Pygame to clean up resources.*
56. pygame.quit()
57. *# Exit the Python program gracefully.*
58. quit()

## Music Player

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

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. *# 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. *# Function to pause the currently playing music*
12. def pause\_music():
13. pygame.mixer.music.pause()  *# Pause the music*
14. *# Function to unpause the currently paused music*
15. def unpause\_music():
16. pygame.mixer.music.unpause()  *# Unpause the music*
17. *# Function to stop the currently playing music*
18. def stop\_music():
19. pygame.mixer.music.stop()  *# Stop the music*
20. *# Function to choose a directory containing music files*
21. def choose\_directory():
22. directory = filedialog.askdirectory()  *# Open a dialog to select a directory*
23. os.chdir(directory)  *# Change the current working directory to the chosen directory*
24. song\_listbox.delete(0, tk.END)  *# Clear the listbox of any previous entries*
25. for song in os.listdir(directory):  *# Loop through files in the directory*
26. if song.endswith(".mp3"):  *# Assuming all songs are in MP3 format*
27. song\_listbox.insert(tk.END, song)  *# Insert the song name into the listbox*
28. *# Create the main application window*
29. root = tk.Tk()
30. root.title("Simple Music Player")  *# Set the title of the window*
31. root.geometry("400x300")  *# Set the initial size of the window*
32. selected\_song = tk.StringVar()  *# Create a variable to store the selected song*
33. song\_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  *# Create a listbox to display song names*
34. song\_listbox.pack(pady=20)  *# Place the listbox in the window with some padding*
35. *# Create buttons for controlling the music*
36. play\_button = tk.Button(root, text="Play", command=play\_music)  *# Play button*
37. play\_button.pack()  *# Place the button in the window*
38. pause\_button = tk.Button(root, text="Pause", command=pause\_music)  *# Pause button*
39. pause\_button.pack()  *# Place the button in the window*
40. unpause\_button = tk.Button(root, text="Unpause", command=unpause\_music)  *# Unpause button*
41. unpause\_button.pack()  *# Place the button in the window*
42. stop\_button = tk.Button(root, text="Stop", command=stop\_music)  *# Stop button*
43. stop\_button.pack()  *# Place the button in the window*
44. choose\_directory\_button = tk.Button(root, text="Choose Directory", command=choose\_directory)  *# Choose directory button*
45. choose\_directory\_button.pack()  *# Place the button in the window*
46. root.mainloop()  *# Start the main event loop to display the window and handle user interactions*

## 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. *# 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. *# Function to pause the currently playing music*
12. def pause\_music():
13. pygame.mixer.music.pause()  *# Pause the music*
14. *# Function to unpause the currently paused music*
15. def unpause\_music():
16. pygame.mixer.music.unpause()  *# Unpause the music*
17. *# Function to stop the currently playing music*
18. def stop\_music():
19. pygame.mixer.music.stop()  *# Stop the music*
20. *# Function to load music from a specific directory*
21. def load\_music\_from\_directory(directory\_path):
22. os.chdir(directory\_path)  *# Change the current working directory to the chosen directory*
23. song\_listbox.delete(0, tk.END)  *# Clear the listbox of any previous entries*
24. for song in os.listdir(directory\_path):  *# Loop through files in the directory*
25. if song.endswith(".mp3"):  *# Assuming all songs are in MP3 format*
26. song\_listbox.insert(tk.END, song)  *# Insert the song name into the listbox*
27. if song\_listbox.size() > 0:  *# If there are songs in the listbox*
28. selected\_song.set(os.path.join(directory\_path, song\_listbox.get(0)))  *# Set the selected song to the first song in the list*
29. *# Create the main application window*
30. root = tk.Tk()
31. root.title("Simple Music Player")  *# Set the title of the window*
32. root.geometry("400x300")  *# Set the initial size of the window*
33. selected\_song = tk.StringVar()  *# Create a variable to store the selected song*
34. song\_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  *# Create a listbox to display song names*
35. song\_listbox.pack(pady=20)  *# Place the listbox in the window with some padding*
36. *# Load music from a specific directory (update this path to your music directory)*
37. load\_music\_from\_directory(r"D:\Documents\Fl new space\++ Rendered Here")
38. *# Function to choose a directory containing music files*
39. def choose\_directory():
40. directory = filedialog.askdirectory()  *# Open a dialog to select a directory*
41. load\_music\_from\_directory(directory)  *# Load the music from the chosen directory*
42. *# Function to handle the selection of a song from the listbox*
43. def on\_song\_selected(event):
44. selected\_song\_index = song\_listbox.curselection()
45. if selected\_song\_index:
46. selected\_song\_name = song\_listbox.get(selected\_song\_index[0])
47. selected\_song\_path = os.path.join(directory\_path.get(), selected\_song\_name)
48. selected\_song.set(selected\_song\_path)
49. *# ... (rest of the code remains the same)*
50. *# Bind the on\_song\_selected function to the <<ListboxSelect>> event*
51. song\_listbox.bind("<<ListboxSelect>>", on\_song\_selected)
52. *# Create the main application window*
53. root = tk.Tk()
54. root.title("Simple Music Player")  *# Set the title of the window*
55. root.geometry("400x300")  *# Set the initial size of the window*
56. selected\_song = tk.StringVar()  *# Create a variable to store the selected song*
57. song\_listbox = tk.Listbox(root, selectmode=tk.SINGLE)  *# Create a listbox to display song names*
58. song\_listbox.pack(pady=20)  *# Place the listbox in the window with some padding*
59. *# Create buttons for controlling the music*
60. play\_button = tk.Button(root, text="Play", command=play\_music)  *# Play button*
61. play\_button.pack()  *# Place the button in the window*
62. pause\_button = tk.Button(root, text="Pause", command=pause\_music)  *# Pause button*
63. pause\_button.pack()  *# Place the button in the window*
64. unpause\_button = tk.Button(root, text="Unpause", command=unpause\_music)  *# Unpause button*
65. unpause\_button.pack()  *# Place the button in the window*
66. stop\_button = tk.Button(root, text="Stop", command=stop\_music)  *# Stop button*
67. stop\_button.pack()  *# Place the button in the window*
68. choose\_directory\_button = tk.Button(root, text="Choose Directory", command=choose\_directory)  *# Choose directory button*
69. choose\_directory\_button.pack()  *# Place the button in the window*
70. root.mainloop()  *# Start the main event loop to display the window and handle user interactions*

## 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. *# Function to play the selected music*
6. def play\_music():
7. pygame.mixer.init()  *# Initialize the mixer module of pygame*
8. pygame.mixer.music.load(selected\_song.get())  *# Load the selected music file*
9. pygame.mixer.music.play()  *# Start playing the music*
10. *# Function to pause the currently playing music*
11. def pause\_music():
12. pygame.mixer.music.pause()  *# Pause the music*
13. *# Function to unpause the currently paused music*
14. def unpause\_music():
15. pygame.mixer.music.unpause()  *# Unpause the music*
16. *# Function to stop the currently playing music*
17. def stop\_music():
18. pygame.mixer.music.stop()  *# Stop the music*
19. *# Create the main application window*
20. root = tk.Tk()
21. root.title("Simple Music Player")  *# Set the title of the window*
22. root.geometry("400x300")  *# Set the initial size of the window*
23. selected\_song = tk.StringVar()  *# Create a variable to store the selected song*
24. selected\_song.set(r"D:\Documents\Fl new space\++ Rendered Here\FunEDM.mp3")  *# Set the selected song to "FunEDM.mp3"*
25. *# Function to load music from a specific directory*
26. def load\_music\_from\_directory(directory\_path):
27. pass  *# Since we don't need to load music from the directory, we simply pass*
28. *# ... (rest of the code remains the same)*
29. *# Create the main application window*
30. root = tk.Tk()
31. root.title("Simple Music Player")  *# Set the title of the window*
32. root.geometry("400x300")  *# Set the initial size of the window*
33. selected\_song = tk.StringVar()  *# Create a variable to store the selected song*
34. 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"*
35. *# ... (rest of the code remains the same)*
36. *# Create buttons for controlling the music*
37. play\_button = tk.Button(root, text="Play", command=play\_music)  *# Play button*
38. play\_button.pack()  *# Place the button in the window*
39. pause\_button = tk.Button(root, text="Pause", command=pause\_music)  *# Pause button*
40. pause\_button.pack()  *# Place the button in the window*
41. unpause\_button = tk.Button(root, text="Unpause", command=unpause\_music)  *# Unpause button*
42. unpause\_button.pack()  *# Place the button in the window*
43. stop\_button = tk.Button(root, text="Stop", command=stop\_music)  *# Stop button*
44. stop\_button.pack()  *# Place the button in the window*
45. 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.

1. *# from ctypes import windll*
2. *# from pathlib import Path*
3. *# Import required modules*
4. import os
5. import ctypes
6. *# Function to list directories and files in the specified path*
7. def list\_directories\_and\_files\_in\_root(path, log\_file):
8. *# Write a heading for directories in the log file*
9. log\_file.write("Directories:\n")
10. *# Loop through all entries (files and directories) in the specified path*
11. for entry in os.listdir(path):
12. *# Check if the entry is a directory*
13. if os.path.isdir(os.path.join(path, entry)):
14. *# Exclude special directories '.' and '..' from the listing*
15. if entry not in ('.', '..'):
16. *# Print the directory name to the console*
17. print(f"Directory: {entry}")
18. *# Write the directory name to the log file*
19. log\_file.write(f"Directory: {entry}\n")
20. *# Write a heading for files in the log file*
21. log\_file.write("\nFiles:\n")
22. *# Loop through all entries (files and directories) in the specified path again*
23. for entry in os.listdir(path):
24. *# Check if the entry is a file*
25. if os.path.isfile(os.path.join(path, entry)):
26. *# Print the file name to the console*
27. print(f"File: {entry}")
28. *# Write the file name to the log file*
29. log\_file.write(f"File: {entry}\n")
30. *# Function to get the path of a special folder using CSIDL*
31. def get\_folder\_path(csidl):
32. *# Create a buffer to store the folder path (use larger buffer size to accommodate longer paths)*
33. buf = ctypes.create\_unicode\_buffer(1024)  *# 1024 characters buffer size*
34. *# Call the Windows API function to get the folder path using CSIDL*
35. ctypes.windll.shell32.SHGetFolderPathW(None, csidl, None, 0, buf)
36. return buf.value
37. *# Function to process default special folders*
38. def process\_default\_folders(log\_file):
39. *# Define an array of folder CSIDLs (constants) along with their descriptions*
40. default\_folder\_constants = [
41. (0x0000, "This is the Desktop folder:"),
42. (0x0005, "This is the Documents folder:"),
43. (0x000E, "This is the Videos folder:"),
44. (0x0027, "This is the Pictures folder:"),
45. (0x000D, "This is the Music folder:"),
46. (0x0019, "This is the Common Desktop folder:")
47. ]
48. *# Loop through the default folders and get their paths using CSIDL*
49. for csidl, description in default\_folder\_constants:
50. try:
51. *# Get the path of the default folder using CSIDL*
52. folder\_path = get\_folder\_path(csidl)
53. *# Write the folder description to the log file*
54. log\_file.write(f"\n\n{description}\n\n")
55. *# Call the function to list directories and files in the specified folder*
56. list\_directories\_and\_files\_in\_root(folder\_path, log\_file)
57. except Exception as e:
58. *# Print an error message if getting the folder path fails*
59. print(f"Failed to get default folder path for CSIDL {csidl}. Error: {e}")
60. *# Main function*
61. def main():
62. *# Get the path of the 'Documents' folder*
63. documents\_folder\_path = os.path.expanduser("~/Documents")
64. *# Print the current directory and the code file's location*
65. print(f"Listing directories and files in {documents\_folder\_path}")
66. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
67. print(f"Current Code File Location: {current\_code\_file\_path}")
68. *# Create the log file name using the current directory path*
69. log\_file\_name = os.path.join(os.path.dirname(current\_code\_file\_path), "bin", "1.txt")
70. *# Check if the 'bin' folder exists, if not, create it*
71. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
72. if not os.path.exists(bin\_folder\_path):
73. os.makedirs(bin\_folder\_path)
74. *# Open the log file in "append" mode (add data to the existing file)*
75. with open(log\_file\_name, "a") as log\_file:
76. *# Call the function to list directories and files in the specified path (Documents folder)*
77. list\_directories\_and\_files\_in\_root(documents\_folder\_path, log\_file)
78. *# Call the new function to iterate through the default folders and append data to the log file*
79. process\_default\_folders(log\_file)
80. *# Print a success message with the log file name*
81. print(f"Successfully logged to {log\_file\_name}")
82. *# Read and print the contents of the log file to the console*
83. print("\nLogged Contents:")
84. with open(log\_file\_name, "r") as read\_log\_file:
85. for line in read\_log\_file:
86. print(line.rstrip())
87. *# Execute the main function if this script is run directly (not imported as a module)*
88. if \_\_name\_\_ == "\_\_main\_\_":
89. main()

## Adding multiple default folder

1. *# Function to process default special folders*
2. def process\_default\_folders(log\_file):
3. *# Define an array of folder CSIDLs (constants) along with their descriptions*
4. all\_csidl\_values = [
5. (0x0000, "CSIDL\_DESKTOP", "The virtual folder representing the Windows desktop, the root of the shell namespace."),
6. (0x0005, "CSIDL\_DOCUMENTS", "This is the Documents folder:"),
7. (0x000E, "CSIDL\_VIDEOS", "This is the Videos folder:"),
8. (0x0027, "CSIDL\_PICTURES", "This is the Pictures folder:"),
9. (0x000D, "CSIDL\_MUSIC", "This is the Music folder:"),
10. (0x0019, "CSIDL\_COMMON\_DESKTOPDIRECTORY", "This is the Common Desktop folder:"),
11. (0x0002, "CSIDL\_PROGRAMS", "The file system directory that contains the user's program groups (which are themselves file system directories)"),
12. (0x11, "CSIDL\_DRIVES", ""),
13. (0x22, "CSIDL\_HISTORY", ""),
14. (0x20, "CSIDL\_INTERNET\_CACHE", ""),
15. (0x12, "CSIDL\_NETWORK", ""),
16. (0x08, "CSIDL\_RECENT", "The file system directory that contains shortcuts to the user's most recently used documents"),
17. (0x02, "CSIDL\_PROGRAMS", ""),
18. (0x2C, "CSIDL\_PROGRAM\_FILES\_COMMONX86", "A folder for 32-bit components that are shared across applications on 64-bit systems"),
19. (0x2B, "CSIDL\_PROGRAM\_FILES\_COMMON", "A folder for components that are shared across applications"),
20. (0x2A, "CSIDL\_PROGRAM\_FILESX86", "The Program Files folder for 32-bit programs on 64-bit systems"),
21. (0x26, "CSIDL\_PROGRAM\_FILES", "The Program Files folder.")
22. ]

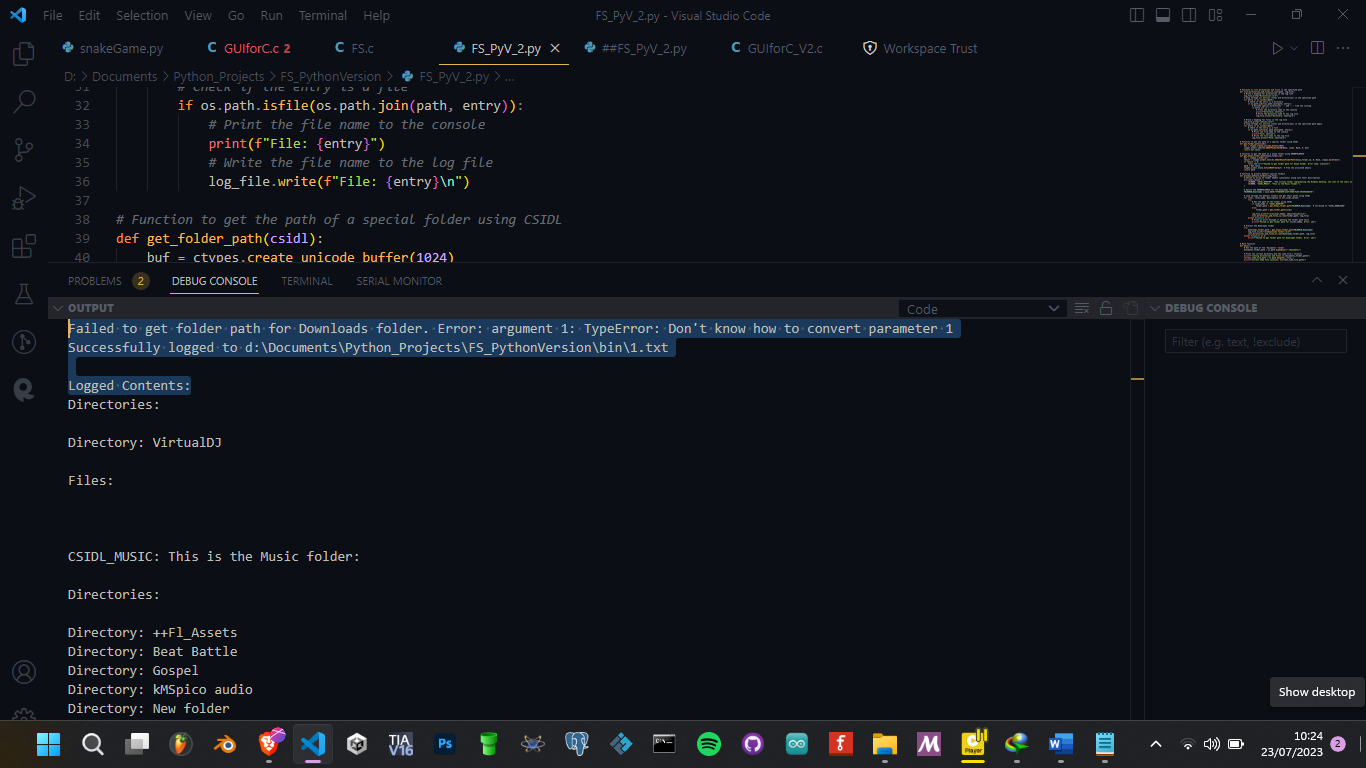
## Trial And Error Adding Downloads Folder

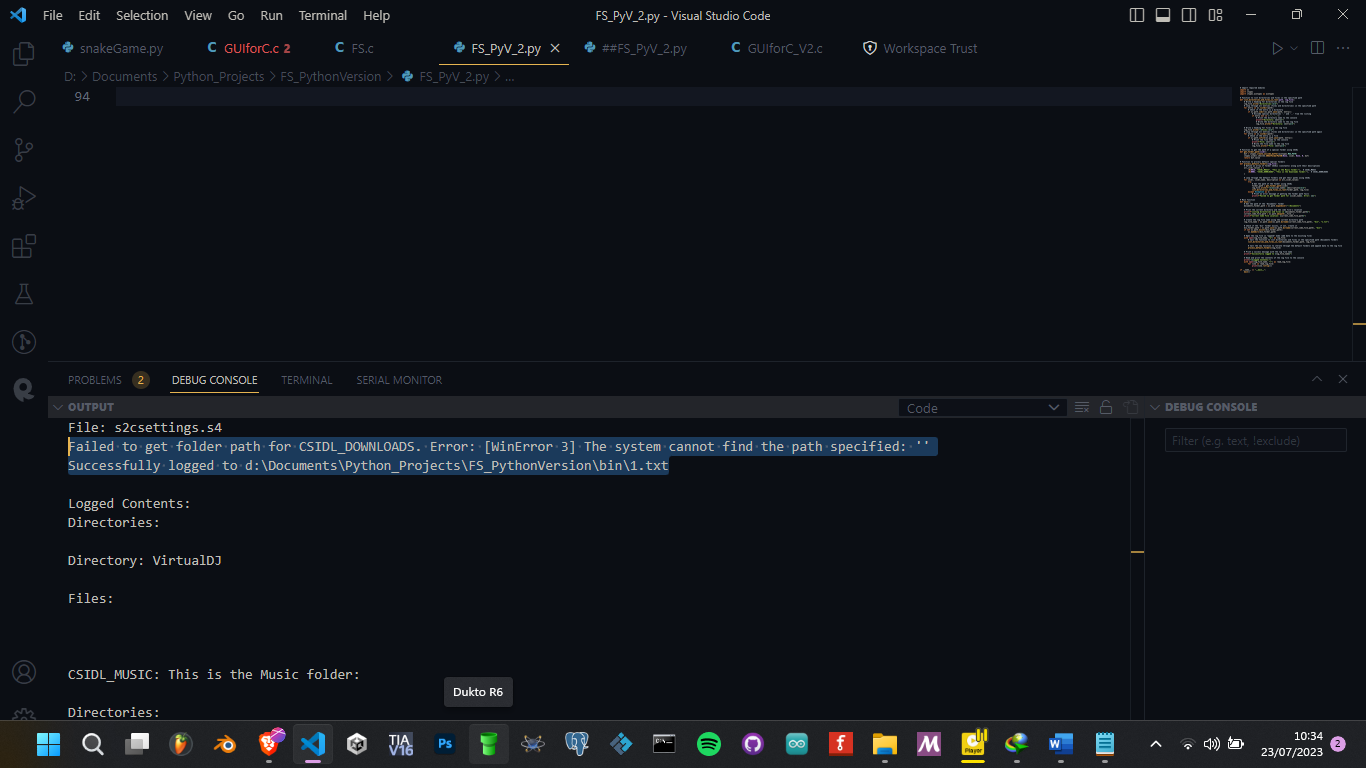
### Accessing Downloads folder using new method

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.**

1. *# import os*
2. *# from ctypes import windll*
3. *# import ctypes*
4. *# from pathlib import Path*
5. *# Import required modules*
6. import os
7. import ctypes
8. import uuid
9. import ctypes.wintypes as wintypes
10. *# Function to list directories and files in the specified path*
11. def list\_directories\_and\_files\_in\_root(path, log\_file):
12. *# Write a heading for directories in the log file*
13. log\_file.write("Directories:\n\n")
14. *# Loop through all entries (files and directories) in the specified path*
15. for entry in os.listdir(path):
16. *# Check if the entry is a directory*
17. if os.path.isdir(os.path.join(path, entry)):
18. *# Exclude special directories '.' and '..' from the listing*
19. if entry not in ('.', '..'):
20. *# Print the directory name to the console*
21. print(f"Directory: {entry}")
22. *# Write the directory name to the log file*
23. log\_file.write(f"Directory: {entry}\n")
24. *# Write a heading for files in the log file*
25. log\_file.write("\nFiles:\n\n")
26. *# Loop through all entries (files and directories) in the specified path again*
27. for entry in os.listdir(path):
28. *# Check if the entry is a file*
29. if os.path.isfile(os.path.join(path, entry)):
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. *# Function to get the path of a special folder using CSIDL*
35. def get\_folder\_path(csidl):
36. buf = ctypes.create\_unicode\_buffer(1024)
37. ctypes.windll.shell32.SHGetFolderPathW(None, csidl, None, 0, buf)
38. return buf.value
39. *# Function to get the path of a known folder using KNOWNFOLDERID*
40. def get\_known\_folder\_path(known\_folder\_id):
41. buf = ctypes.c\_wchar\_p()
42. result = ctypes.windll.shell32.SHGetKnownFolderPath(known\_folder\_id, 0, None, ctypes.byref(buf))
43. if result != 0:
44. raise OSError(f"Failed to get folder path for Known Folder. Error code: {result}")
45. path = buf.value
46. ctypes.windll.ole32.CoTaskMemFree(buf)  *# Free the allocated memory*
47. return path
48. *# Function to process default special folders*
49. def process\_default\_folders(log\_file):
50. *# Define an array of folder CSIDLs (constants) along with their descriptions*
51. all\_csidl\_values = [
52. (0x0000, "CSIDL\_DESKTOP", "The virtual folder representing the Windows desktop, the root of the shell namespace."),
53. (0x000D, "CSIDL\_MUSIC", "This is the Music folder:"),
54. ]
55. *# Define the KNOWNFOLDERID for the Downloads folder*
56. FOLDERID\_Downloads = uuid.UUID("374DE290-123F-4565-9164-39C4925E467B")
57. *# Loop through the default folders and get their paths using CSIDL*
58. for csidl, csidl\_name, description in all\_csidl\_values:
59. try:
60. *# Get the path of the folder using CSIDL*
61. if csidl\_name == "CSIDL\_DESKTOP":
62. folder\_path = get\_known\_folder\_path(FOLDERID\_Downloads)  *# Corrected to "CSIDL\_DOWNLOADS"*
63. else:
64. folder\_path = get\_folder\_path(csidl)
65. log\_file.write(f"\n\n{csidl\_name}: {description}\n\n")
66. list\_directories\_and\_files\_in\_root(folder\_path, log\_file)
67. except Exception as e:
68. *# Print an error message if getting the folder path fails*
69. print(f"Failed to get folder path for {csidl\_name}. Error: {e}")
70. *# Process the Downloads folder*
71. try:
72. downloads\_folder\_path = get\_known\_folder\_path(FOLDERID\_Downloads)
73. log\_file.write("\n\nDownloads folder:\n\n")
74. list\_directories\_and\_files\_in\_root(downloads\_folder\_path, log\_file)
75. except Exception as e:
76. print(f"Failed to get folder path for Downloads folder. Error: {e}")
77. *# Main function*
78. def main():
79. *# Get the path of the 'Documents' folder*
80. documents\_folder\_path = os.path.expanduser("~/Documents")
81. *# Print the current directory and the code file's location*
82. print(f"Listing directories and files in {documents\_folder\_path}")
83. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
84. print(f"Current Code File Location: {current\_code\_file\_path}")
85. *# Create the log file name using the current directory path*
86. log\_file\_name = os.path.join(os.path.dirname(current\_code\_file\_path), "bin", "1.txt")
87. *# Check if the 'bin' folder exists, if not, create it*
88. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
89. if not os.path.exists(bin\_folder\_path):
90. os.makedirs(bin\_folder\_path)
91. *# Open the log file in "append" mode (add data to the existing file)*
92. with open(log\_file\_name, "a") as log\_file:
93. *# Call the function to list directories and files in the specified path (Documents folder)*
94. list\_directories\_and\_files\_in\_root(documents\_folder\_path, log\_file)
95. *# Call the new function to iterate through the default folders and append data to the log file*
96. process\_default\_folders(log\_file)
97. *# Print a success message with the log file name*
98. print(f"Successfully logged to {log\_file\_name}")
99. *# Read and print the contents of the log file to the console*
100. print("\nLogged Contents:")
101. with open(log\_file\_name, "r") as read\_log\_file:
102. for line in read\_log\_file:
103. print(line.rstrip())
104. if \_\_name\_\_ == "\_\_main\_\_":
105. main()



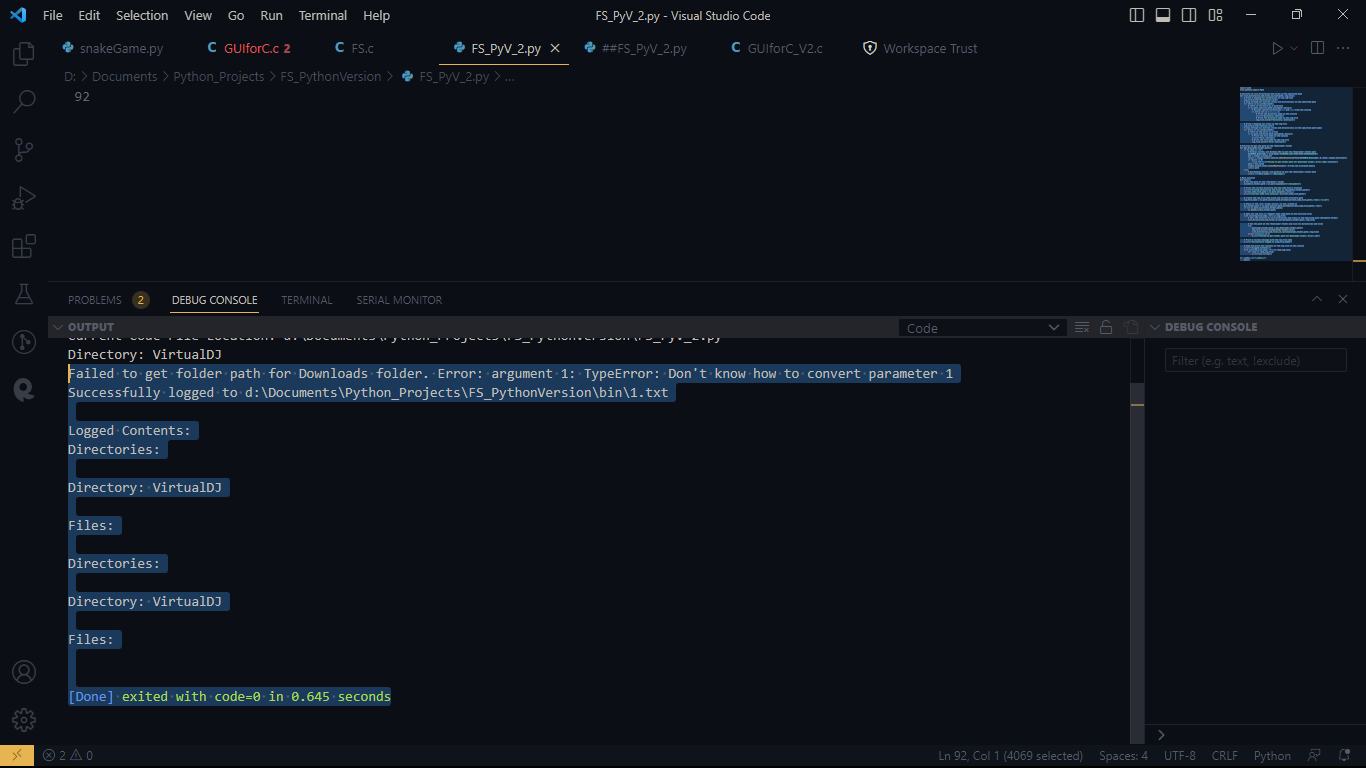


## Accessing multiple operating system Downloads folders

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

1. *# Import required modules*
2. import os
3. import ctypes
4. import ctypes.wintypes as wintypes
5. import uuid
6. from pathlib import Path
7. *# Function to list directories and files in the specified path*
8. def list\_directories\_and\_files\_in\_root(path, log\_file):
9. *# Write a heading for directories in the log file*
10. log\_file.write("Directories:\n\n")
11. *# Loop through all entries (files and directories) in the specified path*
12. for entry in os.listdir(path):
13. *# Check if the entry is a directory*
14. if os.path.isdir(os.path.join(path, entry)):
15. *# Exclude special directories '.' and '..' from the listing*
16. if entry not in ('.', '..'):
17. *# Print the directory name to the console*
18. print(f"Directory: {entry}")
19. *# Write the directory name to the log file*
20. log\_file.write(f"Directory: {entry}\n")
21. *# Write a heading for files in the log file*
22. log\_file.write("\nFiles:\n\n")
23. *# Loop through all entries (files and directories) in the specified path again*
24. for entry in os.listdir(path):
25. *# Check if the entry is a file*
26. if os.path.isfile(os.path.join(path, entry)):
27. *# Print the file name to the console*
28. print(f"File: {entry}")
29. *# Write the file name to the log file*
30. log\_file.write(f"File: {entry}\n")
31. *# Function to get the path of the "Downloads" folder*
32. def get\_downloads\_folder\_path():
33. if os.name == 'nt':
34. *# Windows system, use Windows API to get the "Downloads" folder path*
35. FOLDERID\_Downloads = uuid.UUID("374DE290-123F-4565-9164-39C4925E467B")
36. buf = ctypes.c\_wchar\_p()
37. result = ctypes.windll.shell32.SHGetKnownFolderPath(FOLDERID\_Downloads, 0, None, ctypes.byref(buf))
38. if result != 0:
39. raise OSError(f"Failed to get folder path for Downloads folder. Error code: {result}")
40. path = buf.value
41. ctypes.windll.ole32.CoTaskMemFree(buf)  *# Free the allocated memory*
42. return path
43. else:
44. *# Non-Windows system, use pathlib to get the "Downloads" folder path*
45. return str(Path.home() / "Downloads")
46. *# Main function*
47. def main():
48. *# Get the path of the 'Documents' folder*
49. documents\_folder\_path = os.path.expanduser("~/Documents")
50. *# Print the current directory and the code file's location*
51. print(f"Listing directories and files in {documents\_folder\_path}")
52. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
53. print(f"Current Code File Location: {current\_code\_file\_path}")
54. *# Create the log file name using the current directory path*
55. log\_file\_name = os.path.join(os.path.dirname(current\_code\_file\_path), "bin", "1.txt")
56. *# Check if the 'bin' folder exists, if not, create it*
57. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
58. if not os.path.exists(bin\_folder\_path):
59. os.makedirs(bin\_folder\_path)
60. *# Open the log file in "append" mode (add data to the existing file)*
61. with open(log\_file\_name, "a") as log\_file:
62. *# Call the function to list directories and files in the specified path (Documents folder)*
63. list\_directories\_and\_files\_in\_root(documents\_folder\_path, log\_file)
64. *# Get the path of the "Downloads" folder and list its directories and files*
65. try:
66. downloads\_folder\_path = get\_downloads\_folder\_path()
67. log\_file.write("\n\nDownloads folder:\n\n")
68. list\_directories\_and\_files\_in\_root(downloads\_folder\_path, log\_file)
69. except Exception as e:
70. print(f"Failed to get folder path for Downloads folder. Error: {e}")
71. *# Print a success message with the log file name*
72. print(f"Successfully logged to {log\_file\_name}")
73. *# Read and print the contents of the log file to the console*
74. print("\nLogged Contents:")
75. with open(log\_file\_name, "r") as read\_log\_file:
76. for line in read\_log\_file:
77. print(line.rstrip())
78. if \_\_name\_\_ == "\_\_main\_\_":
79. main()

But it brought the following error:



### 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. 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()

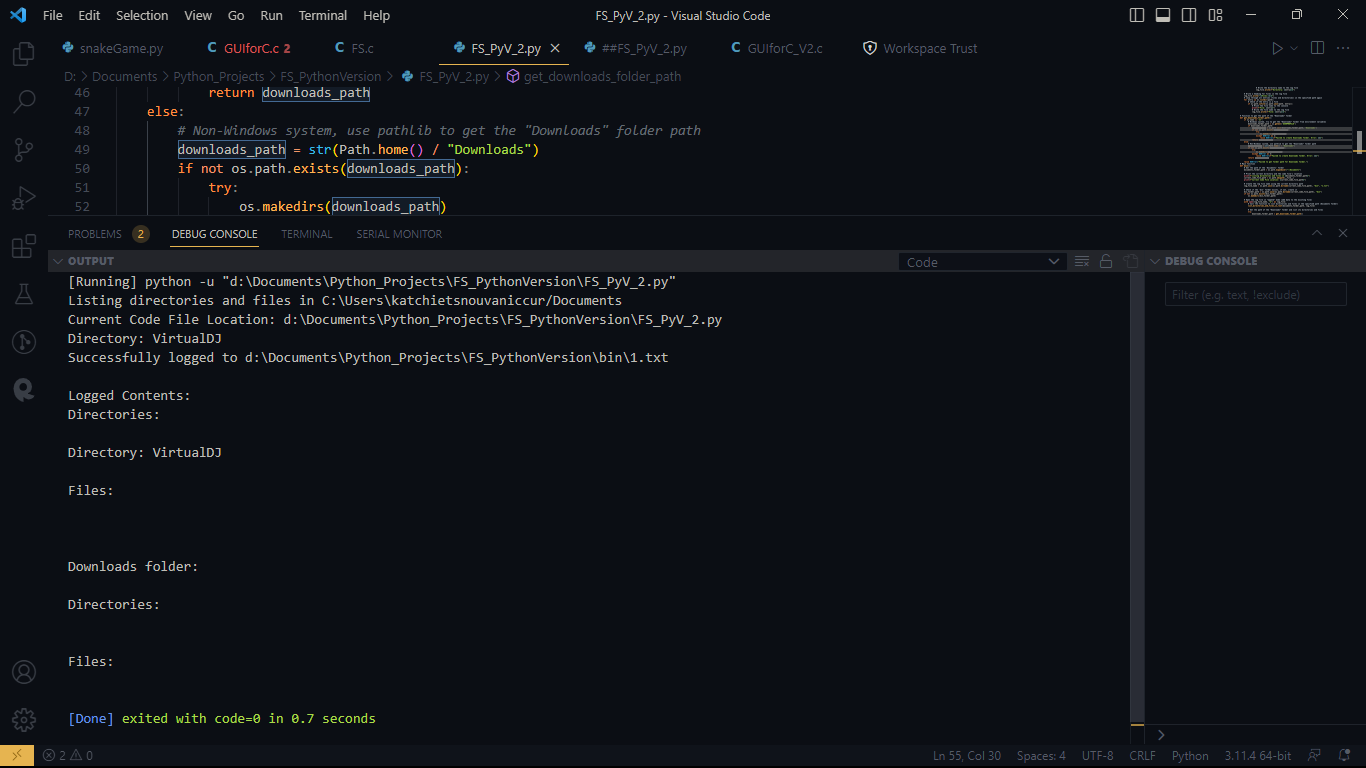
Python will interpret the path correctly when opening the file, and it will work as intended.So, you don't need to worry about the double backslashes in the returned path; it will function correctly in Python for file operations and other functions that require a valid file path.

### 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:
12. raise OSError(f"Failed to create Downloads folder. Error: {e}")
13. return downloads\_path
14. else:
15. *# Non-Windows system, use pathlib to get the "Downloads" folder path*
16. downloads\_path = str(Path.home() / "Downloads")
17. if not os.path.exists(downloads\_path):
18. try:
19. os.makedirs(downloads\_path)
20. except OSError as e:
21. raise OSError(f"Failed to create Downloads folder. Error: {e}")
22. return downloads\_path

Brought nonn error, but old location

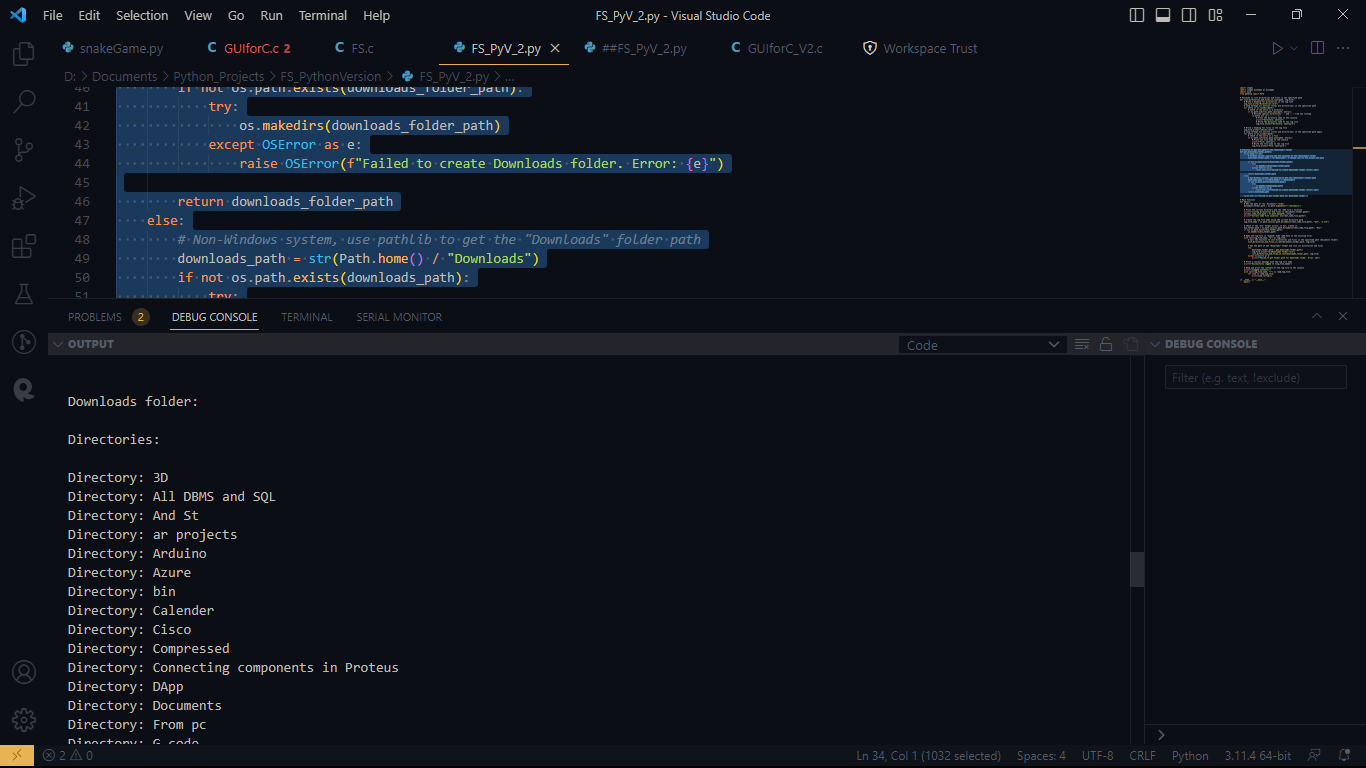


### Try 3: Success, semi hardcoded, specifically for my system

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:

1. *# Function to get the path of the "Downloads" folder*
2. def get\_downloads\_folder\_path():
3. if os.name == 'nt':
4. *# Windows system, specify the new location of the "Downloads" folder*
5. downloads\_folder\_path = "D:/Downloads"  *# Change this to the actual new path*
6. if not os.path.exists(downloads\_folder\_path):
7. try:
8. os.makedirs(downloads\_folder\_path)
9. except OSError as e:
10. raise OSError(f"Failed to create Downloads folder. Error: {e}")
11. return downloads\_folder\_path
12. else:
13. *# Non-Windows system, use pathlib to get the "Downloads" folder path*
14. downloads\_path = str(Path.home() / "Downloads")
15. if not os.path.exists(downloads\_path):
16. try:
17. os.makedirs(downloads\_path)
18. except OSError as e:
19. raise OSError(f"Failed to create Downloads folder. Error: {e}")
20. return downloads\_path
21. 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:

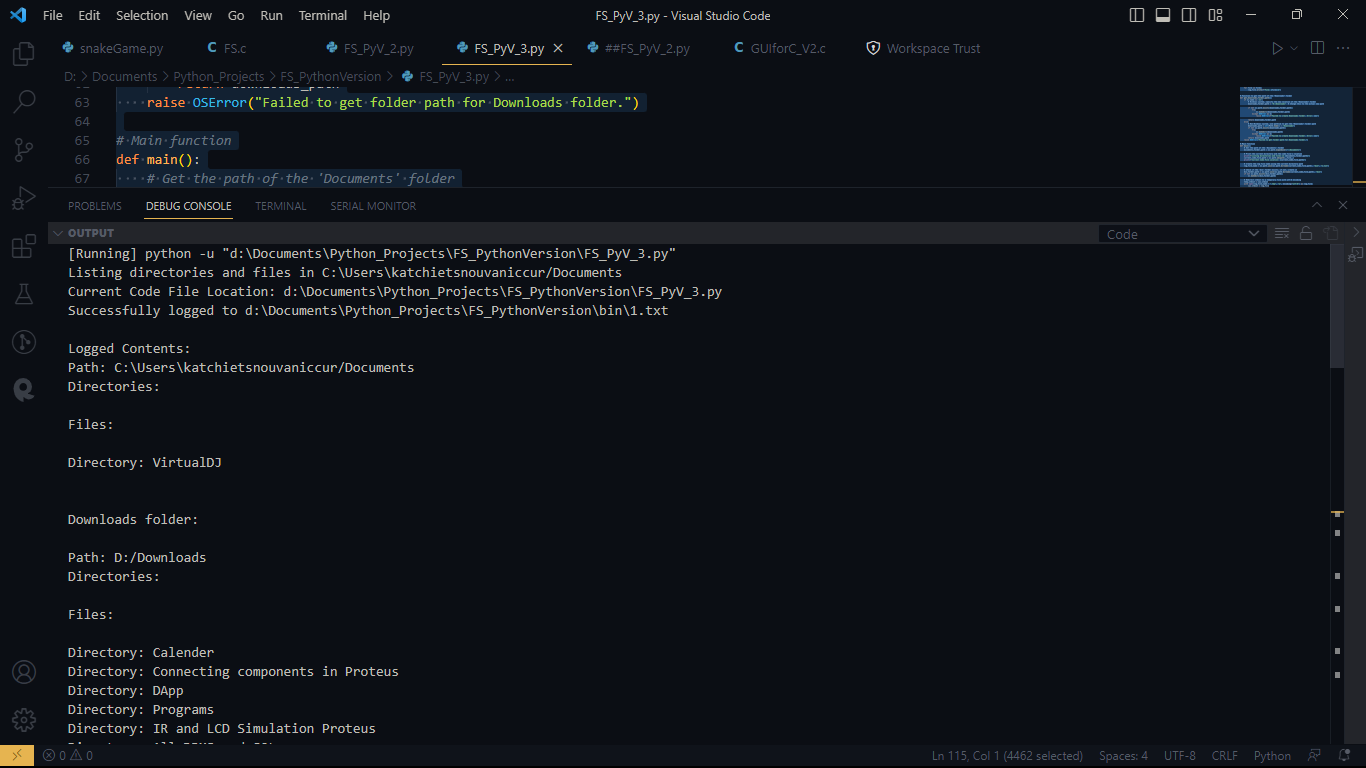
1. import os
2. import ctypes
3. import ctypes.wintypes as wintypes
4. import uuid
5. from pathlib import Path
6. import sys
7. *# Function to list directories and files in the specified path*
8. def list\_directories\_and\_files\_in\_root(path, log\_file):
9. *# Write a heading for directories in the log file*
10. log\_file.write("Directories:\n\n")
11. *# Write a heading for files in the log file*
12. log\_file.write("Files:\n\n")
13. *# Loop through all entries (files and directories) in the specified path*
14. for entry in os.listdir(path):
15. entry\_path = os.path.join(path, entry)
16. *# Check if the entry is a directory*
17. if os.path.isdir(entry\_path):
18. *# Exclude special directories '.' and '..' from the listing*
19. if entry not in ('.', '..'):
20. *# Print the directory name to the console*
21. print(f"Directory: {entry}")
22. *# Write the directory name to the log file*
23. log\_file.write(f"Directory: {entry}\n")
25. *# Check if the entry is a file*
26. elif os.path.isfile(entry\_path):
27. *# Print the file name to the console*
28. print(f"File: {entry}")
29. *# Write the file name to the log file*
30. log\_file.write(f"File: {entry}\n")
31. *# Function to get the path of the "Downloads" folder*
32. def get\_downloads\_folder\_path():
33. if os.name == 'nt':
34. *# Windows system, specify the new location of the "Downloads" folder*
35. downloads\_folder\_path = "D:/Downloads"  *# Change this to the actual new path*
36. if not os.path.exists(downloads\_folder\_path):
37. try:
38. os.makedirs(downloads\_folder\_path)
39. except OSError as e:
40. raise OSError(f"Failed to create Downloads folder. Error: {e}")
41. return downloads\_folder\_path
42. else:
43. *# Non-Windows system, use pathlib to get the "Downloads" folder path*
44. downloads\_path = str(Path.home() / "Downloads")
45. if not os.path.exists(downloads\_path):
46. try:
47. os.makedirs(downloads\_path)
48. except OSError as e:
49. raise OSError(f"Failed to create Downloads folder. Error: {e}")
50. return downloads\_path.encode('utf-8')
51. raise OSError("Failed to get folder path for Downloads folder.")
52. *# Main function*
53. def main():
54. *# Get the path of the 'Documents' folder*
55. documents\_folder\_path = os.path.expanduser("~/Documents")
56. *# Print the current directory and the code file's location*
57. print(f"Listing directories and files in {documents\_folder\_path}")
58. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
59. print(f"Current Code File Location: {current\_code\_file\_path}")
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. *# Check if the 'bin' folder exists, if not, create it*
63. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
64. if not os.path.exists(bin\_folder\_path):
65. os.makedirs(bin\_folder\_path)
66. *# Redirect stdout to a temporary file with utf-8 encoding*
67. temp\_stdout = sys.stdout
68. with open(log\_file\_name + ".tmp", "w", encoding="utf-8") as log\_file:
69. sys.stdout = 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. *# Get the path of the "Downloads" folder and list its directories and files*
73. try:
74. downloads\_folder\_path = get\_downloads\_folder\_path()
75. if downloads\_folder\_path not in (documents\_folder\_path, bin\_folder\_path):
76. print("\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"\n\nFailed to get folder path for Downloads folder. Error: {e}")
80. *# Restore stdout and rename the temporary log file to the final name*
81. sys.stdout = temp\_stdout
82. os.replace(log\_file\_name + ".tmp", log\_file\_name)
83. *# Print a success message with the log file name*
84. print(f"Successfully logged to {log\_file\_name}")
85. *# Read and print the contents of the log file to the console*
86. print("\nLogged Contents:")
87. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
88. for line in read\_log\_file:
89. print(line.rstrip())
90. if \_\_name\_\_ == "\_\_main\_\_":
91. main()

### Try 5

1. This was the most successful one yet:

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

1. *# Import necessary modules from Python's standard library*
2. import os
3. import sys
4. from pathlib import Path
5. *# Function to list directories and files in the specified path*
6. def list\_directories\_and\_files(path, log\_file):
7. *# Write a heading for directories and files in the log file*
8. log\_file.write(f"Path: {path}\n")
9. log\_file.write("Directories:\n\n")
10. log\_file.write("Files:\n\n")
11. *# Initialize two sets to store the names of directories and files*
12. directories = set()
13. files = set()
14. *# Loop through all entries (files and directories) in the specified path*
15. for entry in os.listdir(path):
16. entry\_path = os.path.join(path, entry)
17. *# Check if the entry is a directory*
18. if os.path.isdir(entry\_path):
19. *# Exclude special directories '.' and '..' from the listing*
20. if entry not in ('.', '..'):
21. directories.add(entry)
23. *# Check if the entry is a file*
24. elif os.path.isfile(entry\_path):
25. files.add(entry)
26. *# Write the directories to the log file*
27. for directory in directories:
28. log\_file.write(f"Directory: {directory}\n")
29. *# Write the files to the log file*
30. for file in files:
31. log\_file.write(f"File: {file}\n")
32. *# Function to get the path of the "Downloads" folder*
33. def get\_downloads\_folder\_path():
34. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
35. if os.name == 'nt':  *# Windows system*
36. *# Specify the new location of the "Downloads" folder for Windows*
37. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
38. *# Create the folder if it doesn't exist*
39. if not os.path.exists(downloads\_folder\_path):
40. try:
41. os.makedirs(downloads\_folder\_path)
42. except OSError as e:
43. raise OSError(f"Failed to create Downloads folder. Error: {e}")
44. return downloads\_folder\_path
45. else:  *# Non-Windows system*
46. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
47. downloads\_path = str(Path.home() / "Downloads")
48. *# Create the folder if it doesn't exist*
49. if not os.path.exists(downloads\_path):
50. try:
51. os.makedirs(downloads\_path)
52. except OSError as e:
53. raise OSError(f"Failed to create Downloads folder. Error: {e}")
54. return downloads\_path
55. *# If the code reaches this point, it means the function couldn't determine the Downloads folder path*
56. raise OSError("Failed to get folder path for Downloads folder.")
57. *# Main function*
58. def main():
59. *# Get the path of the 'Documents' folder*
60. documents\_folder\_path = os.path.expanduser("~/Documents")
61. *# Print the current directory and the code file's location*
62. print(f"Listing directories and files in {documents\_folder\_path}")
63. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
64. print(f"Current Code File Location: {current\_code\_file\_path}")
65. *# Create the log file name using the current directory path*
66. log\_file\_name = os.path.join(os.path.dirname(current\_code\_file\_path), "bin", "1.txt")
67. *# Check if the 'bin' folder exists, if not, create it*
68. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
69. if not os.path.exists(bin\_folder\_path):
70. os.makedirs(bin\_folder\_path)
71. *# Redirect stdout to a temporary file with utf-8 encoding*
72. temp\_stdout = sys.stdout
73. with open(log\_file\_name + ".tmp", "w", encoding="utf-8") as log\_file:
74. sys.stdout = log\_file
75. *# Call the function to list directories and files in the "Documents" folder*
76. list\_directories\_and\_files(documents\_folder\_path, log\_file)
77. *# Get the path of the "Downloads" folder and list its directories and files*
78. try:
79. downloads\_folder\_path = get\_downloads\_folder\_path()
80. if downloads\_folder\_path != documents\_folder\_path:
81. log\_file.write("\n\nDownloads folder:\n\n")
82. list\_directories\_and\_files(downloads\_folder\_path, log\_file)
83. except Exception as e:
84. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
85. *# Restore stdout and rename the temporary log file to the final name*
86. sys.stdout = temp\_stdout
87. os.replace(log\_file\_name + ".tmp", log\_file\_name)
88. *# Print a success message with the log file name*
89. print(f"Successfully logged to {log\_file\_name}")
90. *# Read and print the contents of the log file to the console*
91. print("\nLogged Contents:")
92. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
93. for line in read\_log\_file:
94. print(line.rstrip())
95. *# The program execution starts here*
96. if \_\_name\_\_ == "\_\_main\_\_":
97. main()



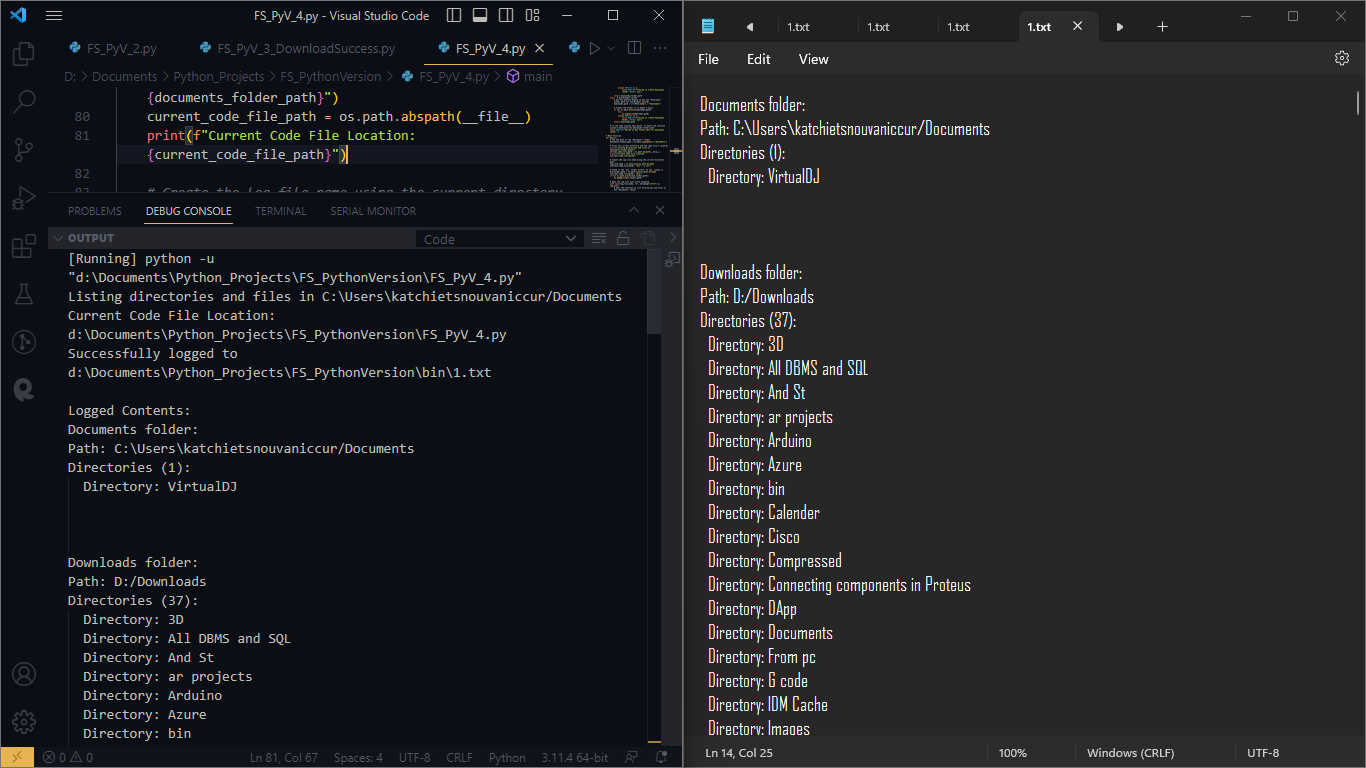
#### To correct a certain grouping issue

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

1. import os
2. import sys
3. from pathlib import Path
4. *# Function to list directories and files in the specified path*
5. def list\_directories\_and\_files(path, log\_file, title):
6. *# Write a heading for directories and files in the log file*
7. log\_file.write(f"{title}:\n")
8. log\_file.write("Directories:\n\n")
9. log\_file.write("Files:\n\n")
10. *# Initialize a set to store the names of directories and files*
11. directories = set()
12. files = set()
13. *# Loop through all entries (files and directories) in the specified path*
14. for entry in os.listdir(path):
15. entry\_path = os.path.join(path, entry)
16. *# Check if the entry is a directory*
17. if os.path.isdir(entry\_path):
18. *# Exclude special directories '.' and '..' from the listing*
19. if entry not in ('.', '..'):
20. directories.add(entry)
22. *# Check if the entry is a file*
23. elif os.path.isfile(entry\_path):
24. files.add(entry)
25. *# Write the directories to the log file*
26. for directory in directories:
27. log\_file.write(f"Directory: {directory}\n")
28. *# Write the files to the log file*
29. for file in files:
30. log\_file.write(f"File: {file}\n")
31. *# Rest of the code remains the same...*
32. *# Main function*
33. def main():
34. *# Rest of the code remains the same...*
35. *# Call the function to list directories and files in the "Documents" folder*
36. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
37. *# Get the path of the "Downloads" folder and list its directories and files*
38. try:
39. downloads\_folder\_path = get\_downloads\_folder\_path()
40. if downloads\_folder\_path != documents\_folder\_path:
41. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder")
42. except Exception as e:
43. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
44. *# Rest of the code remains the same...*
45. if \_\_name\_\_ == "\_\_main\_\_":
46. 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. *# Function to list directories and files in the specified path*
6. def list\_directories\_and\_files(path, log\_file, title):
7. *# Write a heading for directories and files in the log file*
8. log\_file.write(f"{title}:\n")
9. *# Initialize lists to store the names of directories and files*
10. directories = []
11. files = []
12. *# Loop through all entries (files and directories) in the specified path*
13. for entry in os.listdir(path):
14. entry\_path = os.path.join(path, entry)
15. *# Check if the entry is a directory*
16. if os.path.isdir(entry\_path):
17. *# Exclude special directories '.' and '..' from the listing*
18. if entry not in ('.', '..'):
19. directories.append(entry)
21. *# Check if the entry is a file*
22. elif os.path.isfile(entry\_path):
23. files.append(entry)
24. *# Write the directories to the log file*
25. if directories:
26. log\_file.write(f"Path: {path}\n")
27. log\_file.write(f"Directories ({len(directories)}):\n")
28. for directory in directories:
29. log\_file.write(f"  Directory: {directory}\n")
30. *# Write the files to the log file*
31. if files:
32. log\_file.write(f"Path: {path}\n")
33. log\_file.write(f"Files ({len(files)}):\n")
34. for file in files:
35. log\_file.write(f"  File: {file}\n")
36. *# Function to get the path of the "Downloads" folder*
37. def get\_downloads\_folder\_path():
38. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
39. if os.name == 'nt':  *# Windows system*
40. *# Specify the new location of the "Downloads" folder for Windows*
41. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
42. *# Create the folder if it doesn't exist*
43. if not os.path.exists(downloads\_folder\_path):
44. try:
45. os.makedirs(downloads\_folder\_path)
46. except OSError as e:
47. raise OSError(f"Failed to create Downloads folder. Error: {e}")
48. return downloads\_folder\_path
49. else:  *# Non-Windows system*
50. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
51. downloads\_path = str(Path.home() / "Downloads")
52. *# Create the folder if it doesn't exist*
53. if not os.path.exists(downloads\_path):
54. try:
55. os.makedirs(downloads\_path)
56. except OSError as e:
57. raise OSError(f"Failed to create Downloads folder. Error: {e}")
58. return downloads\_path
59. *# If the code reaches this point, it means the function couldn't determine the Downloads folder path*
60. raise OSError("Failed to get folder path for Downloads folder.")
61. *# Main function*
62. def main():
63. *# Get the path of the 'Documents' folder*
64. documents\_folder\_path = os.path.expanduser("~/Documents")
65. *# Print the current directory and the code file's location*
66. print(f"Listing directories and files in {documents\_folder\_path}")
67. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
68. print(f"Current Code File Location: {current\_code\_file\_path}")
69. *# Create the log file name using the current directory path*
70. log\_file\_name = os.path.join(os.path.dirname(current\_code\_file\_path), "bin", "1.txt")
71. *# Check if the 'bin' folder exists, if not, create it*
72. bin\_folder\_path = os.path.join(os.path.dirname(current\_code\_file\_path), "bin")
73. if not os.path.exists(bin\_folder\_path):
74. os.makedirs(bin\_folder\_path)
75. *# Open the log file with utf-8 encoding*
76. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
77. *# Call the function to list directories and files in the "Documents" folder*
78. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
79. *# Add separation for major groups (8-line separation)*
80. log\_file.write("\n" \* 3)
81. *# Get the path of the "Downloads" folder and list its directories and files*
82. try:
83. downloads\_folder\_path = get\_downloads\_folder\_path()
84. if downloads\_folder\_path != documents\_folder\_path:
85. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder")
86. except Exception as e:
87. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
88. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
89. *# Print a success message with the log file name*
90. print(f"Successfully logged to {log\_file\_name}")
91. *# Read and print the contents of the log file to the console*
92. print("\nLogged Contents:")
93. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
94. for line in read\_log\_file:
95. print(line.rstrip())
96. *# The program execution starts here*
97. if \_\_name\_\_ == "\_\_main\_\_":
98. main()

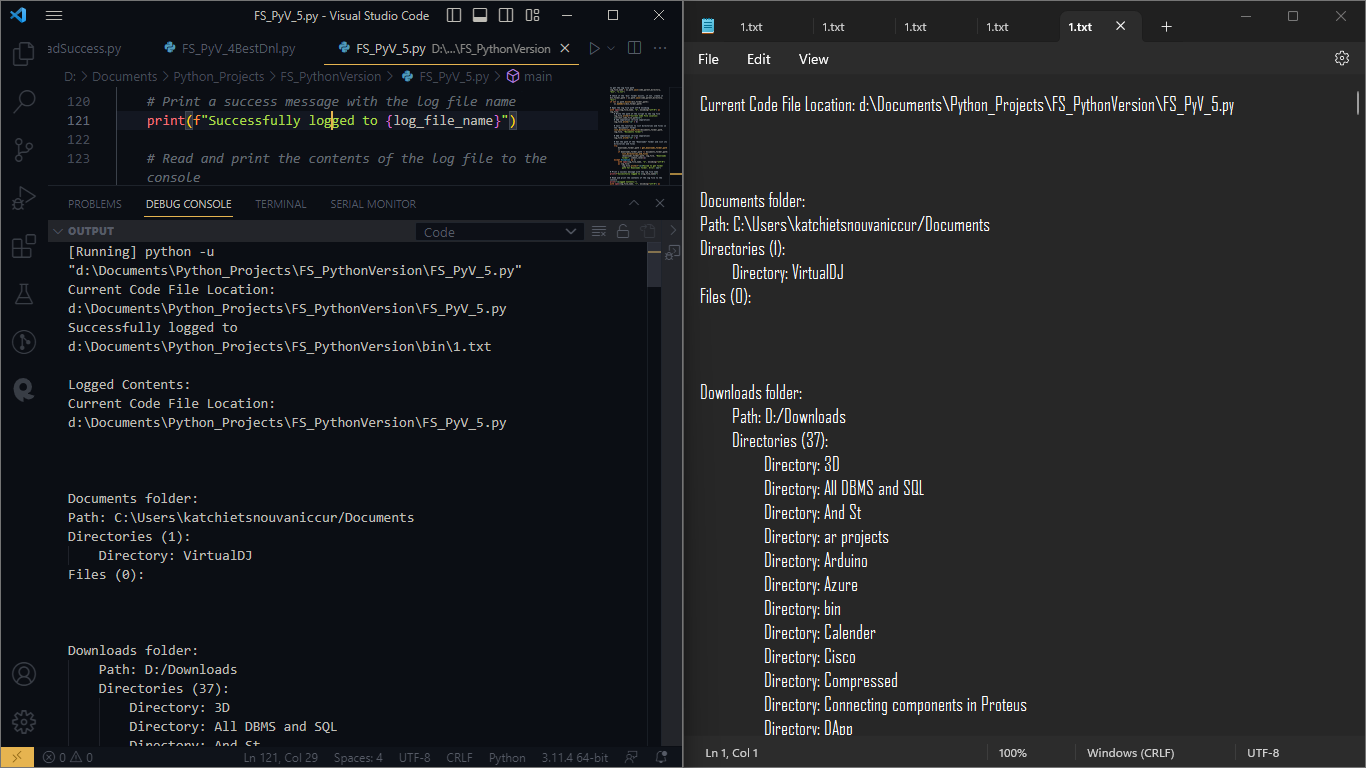


### Try 7: Improvement: Root Storage memory, user interface viewing

Feature to create the "bin" folder relative to the location of the code file.

Also, the logging had improved formatting

1. *# Import necessary modules from Python's standard library*
2. import os
3. import sys
4. from pathlib import Path
5. *# Function to list directories and files in the specified path*
6. def list\_directories\_and\_files(path, log\_file, title, indent\_level=0):
7. *# Write a heading for directories and files in the log file*
8. log\_file.write(f"{title}:\n")
9. indent = "\t" \* indent\_level
10. indent\_for\_files = "\t" \* (indent\_level + 1)
11. *# Initialize lists to store the names of directories and files*
12. directories = []
13. files = []
14. *# Loop through all entries (files and directories) in the specified path*
15. for entry in os.listdir(path):
16. entry\_path = os.path.join(path, entry)
17. *# Check if the entry is a directory*
18. if os.path.isdir(entry\_path):
19. *# Exclude special directories '.' and '..' from the listing*
20. if entry not in ('.', '..'):
21. directories.append(entry)
23. *# Check if the entry is a file*
24. elif os.path.isfile(entry\_path):
25. files.append(entry)
26. *# Write the directories to the log file*
27. log\_file.write(f"{indent}Path: {path}\n")
28. if directories:
29. log\_file.write(f"{indent}Directories ({len(directories)}):\n")
30. for directory in directories:
31. log\_file.write(f"{indent\_for\_files}Directory: {directory}\n")
32. else:
33. log\_file.write(f"{indent}Directories (0):\n")
34. *# Write the files to the log file*
35. if files:
36. log\_file.write(f"{indent}Files ({len(files)}):\n")
37. for file in files:
38. log\_file.write(f"{indent\_for\_files}File: {file}\n")
39. else:
40. log\_file.write(f"{indent}Files (0):\n")
41. *# Function to get the path of the "Downloads" folder*
42. def get\_downloads\_folder\_path():
43. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
44. if os.name == 'nt':  *# Windows system*
45. *# Specify the new location of the "Downloads" folder for Windows*
46. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
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. return downloads\_folder\_path
54. else:  *# Non-Windows system*
55. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
56. downloads\_path = str(Path.home() / "Downloads")
57. *# Create the folder if it doesn't exist*
58. if not os.path.exists(downloads\_path):
59. try:
60. os.makedirs(downloads\_path)
61. except OSError as e:
62. raise OSError(f"Failed to create Downloads folder. Error: {e}")
63. return downloads\_path
64. *# If the code reaches this point, it means the function couldn't determine the Downloads folder path*
65. raise OSError("Failed to get folder path for Downloads folder.")
66. *# Main function*
67. def main():
68. *# Get the path of the 'Documents' folder*
69. documents\_folder\_path = os.path.expanduser("~/Documents")
70. *# Print the current directory and the code file's location*
71. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
72. print(f"Current Code File Location: {current\_code\_file\_path}")
73. *# Create the log file name using the current directory path*
74. *# Get the parent directory of the code file (where the code is located)*
75. code\_parent\_directory = os.path.dirname(current\_code\_file\_path)
76. *# Combine the parent directory with the 'bin' folder name to get the log file path*
77. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
78. *# Check if the 'bin' folder exists, if not, create it*
79. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
80. if not os.path.exists(bin\_folder\_path):
81. os.makedirs(bin\_folder\_path)
82. *# Open the log file with utf-8 encoding*
83. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
84. *# Write the path of the script to the log file*
85. log\_file.write(f"Current Code File Location: {current\_code\_file\_path}\n")
86. *# Add separation (3-line separation)*
87. log\_file.write("\n" \* 3)
88. *# Call the function to list directories and files in the "Documents" folder*
89. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
90. *# Add separation (3-line separation)*
91. log\_file.write("\n" \* 3)
92. *# Get the path of the "Downloads" folder and list its directories and files*
93. try:
94. downloads\_folder\_path = get\_downloads\_folder\_path()
95. if downloads\_folder\_path != documents\_folder\_path:
96. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
97. except Exception as e:
98. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
99. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
100. *# Print a success message with the log file name*
101. print(f"Successfully logged to {log\_file\_name}")
102. *# Read and print the contents of the log file to the console*
103. print("\nLogged Contents:")
104. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
105. for line in read\_log\_file:
106. print(line.rstrip())
107. *# The program execution starts here*
108. if \_\_name\_\_ == "\_\_main\_\_":
109. main()



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/**](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>

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. app = Flask(\_\_name\_\_)
8. *# Function to list directories and files in the specified path*
9. def list\_directories\_and\_files(path, log\_file, title, indent\_level=0):
10. *# Write a heading for directories and files in the log file*
11. log\_file.write(f"{title}:\n")
12. indent = "\t" \* indent\_level
13. indent\_for\_files = "\t" \* (indent\_level + 1)
14. *# Initialize lists to store the names of directories and files*
15. directories = []
16. files = []
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. *# Check if the entry is a directory*
21. if os.path.isdir(entry\_path):
22. *# Exclude special directories '.' and '..' from the listing*
23. if entry not in ('.', '..'):
24. directories.append(entry)
26. *# Check if the entry is a file*
27. elif os.path.isfile(entry\_path):
28. files.append(entry)
29. *# Write the directories to the log file*
30. log\_file.write(f"{indent}Path: {path}\n")
31. if directories:
32. log\_file.write(f"{indent}Directories ({len(directories)}):\n")
33. for directory in directories:
34. log\_file.write(f"{indent\_for\_files}Directory: {directory}\n")
35. else:
36. log\_file.write(f"{indent}Directories (0):\n")
37. *# Write the files to the log file*
38. if files:
39. log\_file.write(f"{indent}Files ({len(files)}):\n")
40. for file in files:
41. log\_file.write(f"{indent\_for\_files}File: {file}\n")
42. else:
43. log\_file.write(f"{indent}Files (0):\n")
44. *# Function to get the path of the "Downloads" folder*
45. def get\_downloads\_folder\_path():
46. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
47. if os.name == 'nt':  *# Windows system*
48. *# Specify the new location of the "Downloads" folder for Windows*
49. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
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:
55. raise OSError(f"Failed to create Downloads folder. Error: {e}")
56. return downloads\_folder\_path
57. else:  *# Non-Windows system*
58. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
59. downloads\_path = str(Path.home() / "Downloads")
60. *# Create the folder if it doesn't exist*
61. if not os.path.exists(downloads\_path):
62. try:
63. os.makedirs(downloads\_path)
64. except OSError as e:
65. raise OSError(f"Failed to create Downloads folder. Error: {e}")
66. return downloads\_path
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. *# Routes for web application*
70. @app.route('/')
71. def index():
72. *# Get the path of the 'Documents' folder*
73. documents\_folder\_path = os.path.expanduser("~/Documents")
74. *# Create the log file name using the current directory path*
75. *# Get the parent directory of the code file (where the code is located)*
76. code\_parent\_directory = os.path.dirname(\_\_file\_\_)
77. *# Combine the parent directory with the 'bin' folder name to get the log file path*
78. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
79. *# Check if the 'bin' folder exists, if not, create it*
80. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
81. if not os.path.exists(bin\_folder\_path):
82. os.makedirs(bin\_folder\_path)
83. *# Open the log file with utf-8 encoding*
84. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
85. *# Write the path of the script to the log file*
86. log\_file.write(f"Current Code File Location: {\_\_file\_\_}\n")
87. *# Add separation (3-line separation)*
88. log\_file.write("\n" \* 3)
89. *# Call the function to list directories and files in the "Documents" folder*
90. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
91. *# Add separation (3-line separation)*
92. log\_file.write("\n" \* 3)
93. *# Get the path of the "Downloads" folder and list its directories and files*
94. try:
95. downloads\_folder\_path = get\_downloads\_folder\_path()
96. if downloads\_folder\_path != documents\_folder\_path:
97. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
98. except Exception as e:
99. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
100. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
101. *# Read the contents of the log file*
102. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
103. log\_contents = read\_log\_file.readlines()
104. return render\_template('index.html', log\_contents=log\_contents)
105. *# Main function*
106. def main():
107. *# Get the path of the 'Documents' folder*
108. documents\_folder\_path = os.path.expanduser("~/Documents")
109. *# Print the current directory and the code file's location*
110. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
111. print(f"Current Code File Location: {current\_code\_file\_path}")
112. *# Create the log file name using the current directory path*
113. *# Get the parent directory of the code file (where the code is located)*
114. code\_parent\_directory = os.path.dirname(current\_code\_file\_path)
115. *# Combine the parent directory with the 'bin' folder name to get the log file path*
116. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
117. *# Check if the 'bin' folder exists, if not, create it*
118. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
119. if not os.path.exists(bin\_folder\_path):
120. os.makedirs(bin\_folder\_path)
121. *# Open the log file with utf-8 encoding*
122. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
123. *# Write the path of the script to the log file*
124. log\_file.write(f"Current Code File Location: {current\_code\_file\_path}\n")
125. *# Add separation (3-line separation)*
126. log\_file.write("\n" \* 3)
127. *# Call the function to list directories and files in the "Documents" folder*
128. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
129. *# Add separation (3-line separation)*
130. log\_file.write("\n" \* 3)
131. *# Get the path of the "Downloads" folder and list its directories and files*
132. try:
133. downloads\_folder\_path = get\_downloads\_folder\_path()
134. if downloads\_folder\_path != documents\_folder\_path:
135. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
136. except Exception as e:
137. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
138. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
139. *# Print a success message with the log file name*
140. print(f"Successfully logged to {log\_file\_name}")
141. *# Read and print the contents of the log file to the console*
142. print("\nLogged Contents:")
143. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
144. for line in read\_log\_file:
145. print(line.rstrip().encode(sys.stdout.encoding, errors='replace').decode(sys.stdout.encoding))
146. if \_\_name\_\_ == "\_\_main\_\_":
147. with ThreadPoolExecutor(max\_workers=2) as executor:
148. executor.submit(main)
149. 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

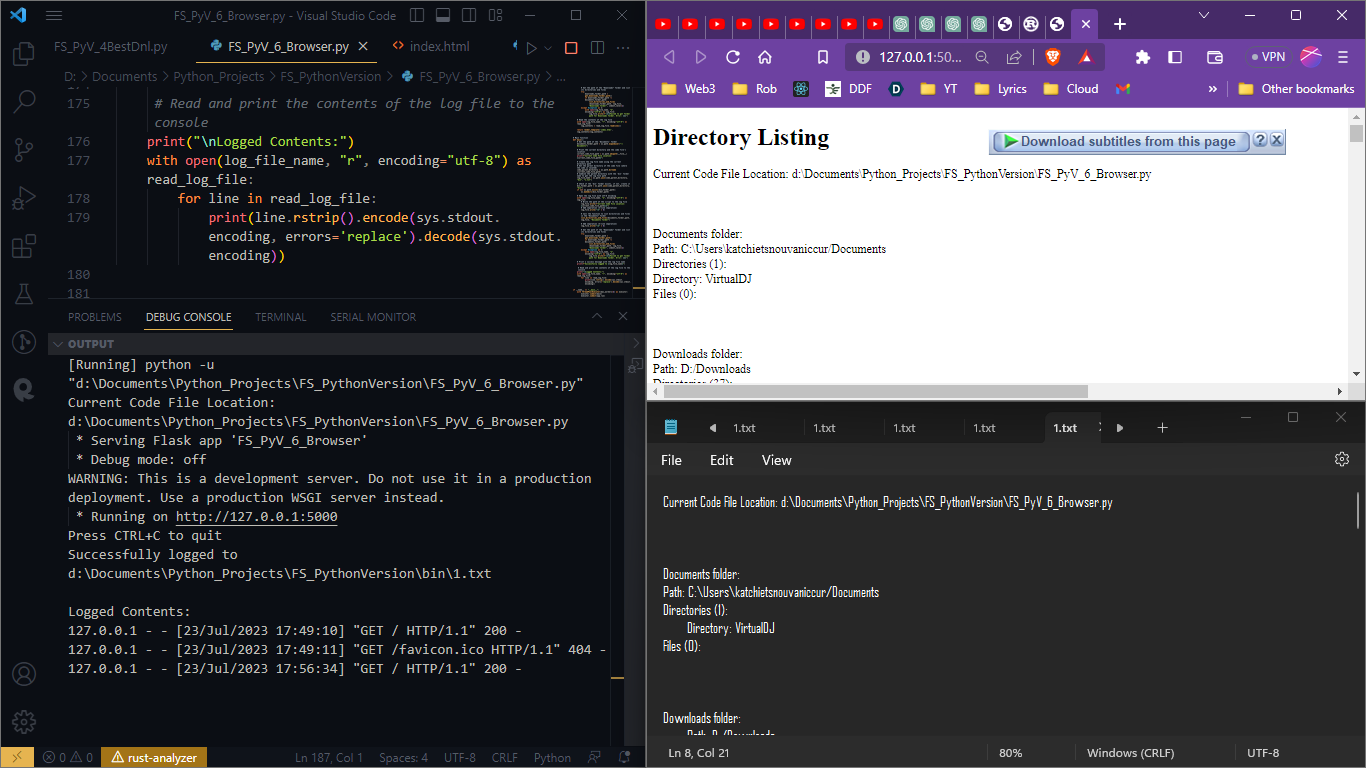
Successfully logged to d:\Documents\Python\_Projects\FS\_PythonVersion\bin\1.txt

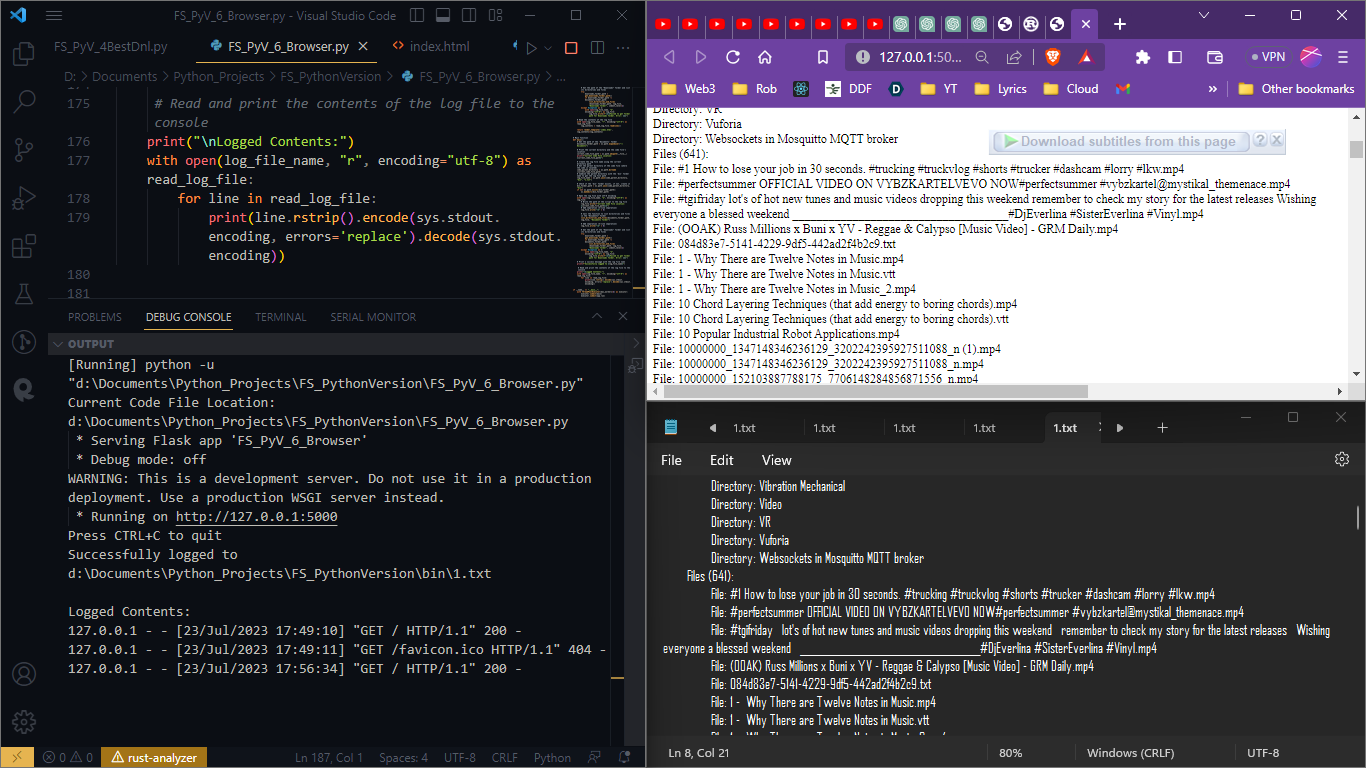
Logged Contents:

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. # Your functions...
5. # Create the Flask app using a function
6. def create\_app():
7. app = Flask(\_\_name\_\_)
8. # Your routes and other app configuration here...
9. # Routes for web application
10. @app.route('/')
11. def index():
12. # ... your existing code ...
13. return app
14. # Main function (if needed)
15. def main():
16. # ... your existing code ...
17. # The program execution starts here
18. if \_\_name\_\_ == "\_\_main\_\_":
19. with ThreadPoolExecutor(max\_workers=2) as executor:
20. executor.submit(main)
21. executor.submit(create\_app().run)

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:
3. from flask import Flask, render\_template
4. import os
5. from pathlib import Path
6. *# Your functions...*
7. *# Create the Flask app using a function*
8. def create\_app():
9. app = Flask(\_\_name\_\_)
10. *# Your routes and other app configuration here...*
11. *# Routes for web application*
12. @app.route('/')
13. def index():
14. *# ... your existing code ...*
15. return app
16. *# Your other functions and main function...*
17. *# The program execution starts here*
18. if \_\_name\_\_ == "\_\_main\_\_":
19. main()
20. Save the changes to the script.
21. 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](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](http://localhost:8080/) or [http://127.0.0.1:8080](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](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](http://0.0.0.0:8080/)" message again.

1. 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.
2. 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**

1. 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.
2. 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
2. *# Import necessary modules from Python's standard library*
3. *# from concurrent.futures import ThreadPoolExecutor*
4. from flask import Flask, render\_template
5. import os
6. from pathlib import Path
7. import sys
8. *# Create the Flask app using a function*
9. def create\_app():
10. app = Flask(\_\_name\_\_)
11. *# Your routes and other app configuration here...*
12. *# Routes for web application*
13. @app.route('/')
14. def index():
15. *# Get the path of the 'Documents' folder*
16. documents\_folder\_path = os.path.expanduser("~/Documents")
17. *# Create the log file name using the current directory path*
18. *# Get the parent directory of the code file (where the code is located)*
19. code\_parent\_directory = os.path.dirname(\_\_file\_\_)
20. *# Combine the parent directory with the 'bin' folder name to get the log file path*
21. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
22. *# Check if the 'bin' folder exists, if not, create it*
23. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
24. if not os.path.exists(bin\_folder\_path):
25. os.makedirs(bin\_folder\_path)
26. *# Open the log file with utf-8 encoding*
27. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
28. *# Write the path of the script to the log file*
29. log\_file.write(f"Current Code File Location: {\_\_file\_\_}\n")
30. *# Add separation (3-line separation)*
31. log\_file.write("\n" \* 3)
32. *# Call the function to list directories and files in the "Documents" folder*
33. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
34. *# Add separation (3-line separation)*
35. log\_file.write("\n" \* 3)
36. *# Get the path of the "Downloads" folder and list its directories and files*
37. try:
38. downloads\_folder\_path = get\_downloads\_folder\_path()
39. if downloads\_folder\_path != documents\_folder\_path:
40. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
41. except Exception as e:
42. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
43. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
44. *# Read the contents of the log file*
45. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
46. log\_contents = read\_log\_file.readlines()
47. return render\_template('index.html', log\_contents=log\_contents)
48. return app
50. *# Function to list directories and files in the specified path*
51. def list\_directories\_and\_files(path, log\_file, title, indent\_level=0):
52. *# Write a heading for directories and files in the log file*
53. log\_file.write(f"{title}:\n")
54. indent = "\t" \* indent\_level
55. indent\_for\_files = "\t" \* (indent\_level + 1)
56. *# Initialize lists to store the names of directories and files*
57. directories = []
58. files = []
59. *# Loop through all entries (files and directories) in the specified path*
60. for entry in os.listdir(path):
61. entry\_path = os.path.join(path, entry)
62. *# Check if the entry is a directory*
63. if os.path.isdir(entry\_path):
64. *# Exclude special directories '.' and '..' from the listing*
65. if entry not in ('.', '..'):
66. directories.append(entry)
68. *# Check if the entry is a file*
69. elif os.path.isfile(entry\_path):
70. files.append(entry)
71. *# Write the directories to the log file*
72. log\_file.write(f"{indent}Path: {path}\n")
73. if directories:
74. log\_file.write(f"{indent}Directories ({len(directories)}):\n")
75. for directory in directories:
76. log\_file.write(f"{indent\_for\_files}Directory: {directory}\n")
77. else:
78. log\_file.write(f"{indent}Directories (0):\n")
79. *# Write the files to the log file*
80. if files:
81. log\_file.write(f"{indent}Files ({len(files)}):\n")
82. for file in files:
83. log\_file.write(f"{indent\_for\_files}File: {file}\n")
84. else:
85. log\_file.write(f"{indent}Files (0):\n")
86. *# Function to get the path of the "Downloads" folder*
87. def get\_downloads\_folder\_path():
88. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
89. if os.name == 'nt':  *# Windows system*
90. *# Specify the new location of the "Downloads" folder for Windows*
91. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
92. *# Create the folder if it doesn't exist*
93. if not os.path.exists(downloads\_folder\_path):
94. try:
95. os.makedirs(downloads\_folder\_path)
96. except OSError as e:
97. raise OSError(f"Failed to create Downloads folder. Error: {e}")
98. return downloads\_folder\_path
99. else:  *# Non-Windows system*
100. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
101. downloads\_path = str(Path.home() / "Downloads")
102. *# Create the folder if it doesn't exist*
103. if not os.path.exists(downloads\_path):
104. try:
105. os.makedirs(downloads\_path)
106. except OSError as e:
107. raise OSError(f"Failed to create Downloads folder. Error: {e}")
108. return downloads\_path
109. *# If the code reaches this point, it means the function couldn't determine the Downloads folder path*
110. raise OSError("Failed to get folder path for Downloads folder.")
112. *# Main function*
113. def main():
114. *# Get the path of the 'Documents' folder*
115. documents\_folder\_path = os.path.expanduser("~/Documents")
116. *# Print the current directory and the code file's location*
117. current\_code\_file\_path = os.path.abspath(\_\_file\_\_)
118. print(f"Current Code File Location: {current\_code\_file\_path}")
119. *# Create the log file name using the current directory path*
120. *# Get the parent directory of the code file (where the code is located)*
121. code\_parent\_directory = os.path.dirname(current\_code\_file\_path)
122. *# Combine the parent directory with the 'bin' folder name to get the log file path*
123. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
124. *# Check if the 'bin' folder exists, if not, create it*
125. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
126. if not os.path.exists(bin\_folder\_path):
127. os.makedirs(bin\_folder\_path)
128. *# Open the log file with utf-8 encoding*
129. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
130. *# Write the path of the script to the log file*
131. log\_file.write(f"Current Code File Location: {current\_code\_file\_path}\n")
132. *# Add separation (3-line separation)*
133. log\_file.write("\n" \* 3)
134. *# Call the function to list directories and files in the "Documents" folder*
135. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
136. *# Add separation (3-line separation)*
137. log\_file.write("\n" \* 3)
138. *# Get the path of the "Downloads" folder and list its directories and files*
139. try:
140. downloads\_folder\_path = get\_downloads\_folder\_path()
141. if downloads\_folder\_path != documents\_folder\_path:
142. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
143. except Exception as e:
144. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
145. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
146. *# Print a success message with the log file name*
147. print(f"Successfully logged to {log\_file\_name}")
148. *# Read and print the contents of the log file to the console*
149. print("\nLogged Contents:")
150. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
151. for line in read\_log\_file:
152. print(line.rstrip().encode(sys.stdout.encoding, errors='replace').decode(sys.stdout.encoding))
153. *# The program execution starts here*
154. if \_\_name\_\_ == "\_\_main\_\_":
155. 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**

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

**python app.py**

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

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

1. 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.
2. *# Import necessary modules from Python's standard library*
3. from flask import Flask, render\_template
4. import os
5. from pathlib import Path
6. from waitress import serve
7. *# Create the Flask app using a function*
8. def create\_app():
9. app = Flask(\_\_name\_\_)
10. *# Your routes and other app configuration here...*
11. *# Routes for web application*
12. @app.route('/')
13. def index():
14. *# Get the path of the 'Documents' folder*
15. documents\_folder\_path = os.path.expanduser("~/Documents")
16. *# Create the log file name using the current directory path*
17. *# Get the parent directory of the code file (where the code is located)*
18. code\_parent\_directory = os.path.dirname(\_\_file\_\_)
19. *# Combine the parent directory with the 'bin' folder name to get the log file path*
20. log\_file\_name = os.path.join(code\_parent\_directory, "bin", "1.txt")
21. *# Check if the 'bin' folder exists, if not, create it*
22. bin\_folder\_path = os.path.join(code\_parent\_directory, "bin")
23. if not os.path.exists(bin\_folder\_path):
24. os.makedirs(bin\_folder\_path)
25. *# Open the log file with utf-8 encoding*
26. with open(log\_file\_name, "w", encoding="utf-8") as log\_file:
27. *# Write the path of the script to the log file*
28. log\_file.write(f"Current Code File Location: {\_\_file\_\_}\n")
29. *# Add separation (3-line separation)*
30. log\_file.write("\n" \* 3)
31. *# Call the function to list directories and files in the "Documents" folder*
32. list\_directories\_and\_files(documents\_folder\_path, log\_file, "Documents folder")
33. *# Add separation (3-line separation)*
34. log\_file.write("\n" \* 3)
35. *# Get the path of the "Downloads" folder and list its directories and files*
36. try:
37. downloads\_folder\_path = get\_downloads\_folder\_path()
38. if downloads\_folder\_path != documents\_folder\_path:
39. list\_directories\_and\_files(downloads\_folder\_path, log\_file, "Downloads folder", indent\_level=1)
40. except Exception as e:
41. with open(log\_file\_name, "a", encoding="utf-8") as log\_file:
42. log\_file.write(f"\n\nFailed to get folder path for Downloads folder. Error: {e}")
43. *# Read the contents of the log file*
44. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
45. log\_contents = read\_log\_file.readlines()
46. return render\_template('index.html', log\_contents=log\_contents)
47. return app
48. *# Function to list directories and files in the specified path*
49. def list\_directories\_and\_files(path, log\_file, title, indent\_level=0):
50. *# Write a heading for directories and files in the log file*
51. log\_file.write(f"{title}:\n")
52. indent = "\t" \* indent\_level
53. indent\_for\_files = "\t" \* (indent\_level + 1)
54. *# Initialize lists to store the names of directories and files*
55. directories = []
56. files = []
57. *# Loop through all entries (files and directories) in the specified path*
58. for entry in os.listdir(path):
59. entry\_path = os.path.join(path, entry)
60. *# Check if the entry is a directory*
61. if os.path.isdir(entry\_path):
62. *# Exclude special directories '.' and '..' from the listing*
63. if entry not in ('.', '..'):
64. directories.append(entry)
66. *# Check if the entry is a file*
67. elif os.path.isfile(entry\_path):
68. files.append(entry)
69. *# Write the directories to the log file*
70. log\_file.write(f"{indent}Path: {path}\n")
71. if directories:
72. log\_file.write(f"{indent}Directories ({len(directories)}):\n")
73. for directory in directories:
74. log\_file.write(f"{indent\_for\_files}Directory: {directory}\n")
75. else:
76. log\_file.write(f"{indent}Directories (0):\n")
77. *# Write the files to the log file*
78. if files:
79. log\_file.write(f"{indent}Files ({len(files)}):\n")
80. for file in files:
81. log\_file.write(f"{indent\_for\_files}File: {file}\n")
82. else:
83. log\_file.write(f"{indent}Files (0):\n")
84. *# Function to get the path of the "Downloads" folder*
85. def get\_downloads\_folder\_path():
86. *# Check if the current system is Windows or non-Windows (like macOS, Linux)*
87. if os.name == 'nt':  *# Windows system*
88. *# Specify the new location of the "Downloads" folder for Windows*
89. downloads\_folder\_path = "D:/Downloads"  *# Change this to your actual new path on Windows*
90. *# Create the folder if it doesn't exist*
91. if not os.path.exists(downloads\_folder\_path):
92. try:
93. os.makedirs(downloads\_folder\_path)
94. except OSError as e:
95. raise OSError(f"Failed to create Downloads folder. Error: {e}")
96. return downloads\_folder\_path
97. else:  *# Non-Windows system*
98. *# Use the pathlib module to get the "Downloads" folder path for non-Windows systems*
99. downloads\_path = str(Path.home() / "Downloads")
100. *# Create the folder if it doesn't exist*
101. if not os.path.exists(downloads\_path):
102. try:
103. os.makedirs(downloads\_path)
104. except OSError as e:
105. raise OSError(f"Failed to create Downloads folder. Error: {e}")
106. return downloads\_path
107. *# If the code reaches this point, it means the function couldn't determine the Downloads folder path*
108. raise OSError("Failed to get folder path for Downloads folder.")
109. *# Main function*
110. def main():
111. *# Create the Flask app using the 'create\_app()' function*
112. app = create\_app()
113. *# Specify the host and port for Waitress to listen on*
114. host = '0.0.0.0'  *# Listen on all available network interfaces*
115. port = 8080
116. *# Use Waitress to serve the app*
117. serve(app, host=host, port=port)
118. *# The program execution starts here*
119. if \_\_name\_\_ == "\_\_main\_\_":
120. main()

### File paths

NB look at difference btn

1. *# Get the path of the 'Documents' folder*
2. documents\_folder\_path = os.path.expanduser("~/Documents")

and

1. *# Get the path of the 'Downloads' folder*
2. 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:

1. *# Read and print the contents of the log file to the console*
2. print("\nLogged Contents:")
3. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
4. for line in read\_log\_file:
5. print(line.rstrip())

To:

1. *# Read and print the contents of the log file to the console*
2. print("\nLogged Contents:")
3. with open(log\_file\_name, "r", encoding="utf-8") as read\_log\_file:
4. for line in read\_log\_file:
5. print(line.rstrip().encode(sys.stdout.encoding, errors='replace').decode(sys.stdout.encoding))