Title: Folder System Algorithm,

Author: Shaun Clarke  
Goal: This program mimics some of the basic functionalities of a folder system.

Steps

1. Import Dependencies

* from typing import List, Union, Optional
* from folder import Folder
* from input\_handler import HandleInput
* from menu import Menu
* import sys

2. Define the Folder Class

* Represents a folder with basic actions.
* Constructor:
  + Takes folder\_name as input.
  + Initializes:
    - folder\_name: the name of the folder.
    - files: a list to store file names.
    - sub\_folders: a list to store subfolder objects.
* \_\_eq\_\_(self, other\_folder) -> bool:
  + Returns True if the folder names are equal.
* does\_folder\_exist(self, folder\_name: str) -> bool:
  + Loops through sub\_folders to check if the folder name exists.
  + Returns True if found, otherwise False.
* does\_file\_exist(self, file\_name: str) -> bool:
  + Checks if the file exists in the current folder.
  + Returns True or False.
* add\_folder(self, folder\_name: str) -> str:
  + If folder already exists, returns "folder exists".
  + Otherwise:
    - Creates a Folder object.
    - Appends it to sub\_folders.
    - Verifies the addition and returns "folder added" or "folder not added".
* select\_folder(self, folder\_name: str) -> object:
  + Uses recursion to locate the folder by name.
  + Returns the folder if found.
  + If folder name is "root", returns the root folder.
  + Returns None if not found.
* add\_file\_to\_folder(self, file\_name: str) -> Union[bool, str]:
  + If file exists, returns "file exist".
  + Otherwise adds it and confirms.
  + Returns True if successful, False otherwise.
* \_\_count\_files(self) -> int:
  + Recursively counts files in current and all subfolders.
  + Returns total count as int.
* \_\_len\_\_(self) -> int:
  + Returns result of \_\_count\_files().
  + Enables use of len(folder) syntax.
* \_\_tree\_view(self, prefix="") -> str:
  + Builds a visual tree string showing folder and file layout recursively.
* \_\_str\_\_(self) -> str:
  + Calls \_\_tree\_view() and returns its string.
  + Enables printing a Folder object visually.

3. Define the HandleInput Class

* Handles user input and interacts with the Folder class.
* get\_input(input\_message: str) -> str:
  + Validates and returns non-empty input from user.
* handle\_add\_folder(folder\_object: Folder) -> str:
  + Prompts for folder name.
  + Adds folder and handles errors if it already exists.
* handle\_select\_folder(folder\_object: Folder) -> object:
  + Prompts for folder name.
  + Uses select\_folder() to find and return it.
* handle\_add\_file\_to\_folder(folder\_object: Folder) -> Union[bool, str]:
  + Prompts for file name.
  + Adds the file to folder and returns result.
* handle\_print\_folder(folder\_object: Folder) -> None:
  + Prints the folder’s structure using \_\_str\_\_.
* handle\_count\_files(folder\_object: Folder) -> None:
  + Uses len() to count files and prints result.

4. Define the Menu Class

* Displays the main menu and gets valid input.
* display\_menu(current\_folder: str = False) -> int:
  + Prints folder name and menu options.
  + Calls get\_menu\_number\_input() to get selection.
* get\_menu\_number\_input(input\_message: str, menu\_options: List) -> int:
  + Ensures user inputs a valid number in range.

5. Define the Main Class

* Initializes and controls the application loop.
* Constructor:
  + Accepts Folder, HandleInput, and Menu instances.
  + Sets up root\_folder, active\_folder, and active\_folder\_name.
* main() method:
  + Creates root folder.
  + Loops through menu selection:
    - If Option 1: Add file
    - If Option 2 : Add subfolder
    - If Option 3 : Select subfolder
    - If Option 4 : Print folder
    - If Option 5 : Count files
    - If Option 6 : Exit the program