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