Unit Testing Of Task 2

Unit Testing of `display_csv_as_table`

The `display_csv_as_table` function accepts a file path as input. Therefore, among {int, float, list, string}, the first three are invalid types of file paths. I create one test for each of these types and assert that the function returns an error or an appropriate message indicating that the input is not a valid file path.

Now turning to valid inputs. The input domain can be divided into several equivalence classes (EC):

1. The file name is an empty string.
2. The file name is a non-empty string.
3. The file extension has different cases.
4. The file is a non-CSV file.
5. The file has unaligned rows.
6. The file has only a header.
7. The file has only one row.
8. The file has multiple rows.

The coverage criteria are:

- For each EC, we need to test at least one input.

This leads to the following test requirements:
R1. If the input is in EC1, the function should raise a `FileNotFoundError`.
R2. If the input is in EC2, the function should display the table content, regardless of whether the non-empty string contains special characters, spaces, etc.
R3. If the input is in EC3, the function should display the table content.
R4. If the input is in EC4, the function should display non-CSV table content.
R5. If the input is in EC5, the function should display the unaligned content.
R6. If the input is in EC6, the function should display only the header.
R7. If the input is in EC7, the function should display only a single row.
R8. If the input is in EC8, the function should display multiple rows.

This leads to the corresponding test cases:
TC1. Copy the original file and name it ""
TC2. Copy the original file and name it "products_copy.csv", "products copy.csv", "products@copy.csv" or "PROdUCTS.csv"
TC3. Copy the original file and name it "products.CSV"
TC4. Create a file "non_csv_file.txt" and input "Non-CSV content"
TC5. Create a file "unaligned_rows.csv" and input "Invalid,Data,Format\n1,2,3,4"
TC6. Create a file "only_header.csv" and input "Product,Price,Units"

TC7. Create a file "one_row.csv" and input "Apple,2,10"
TC8. Create a file "multiple_rows.csv" and input "Product,Price,Units\nApple,2,10\nBanana,1,5"

Unit Testing of `display_filtered_table`

The `display_filtered_table` function accepts a file path and a search string as input. Therefore, among various types, non-string types for the file path or search string are invalid. I create one test for each of these types and assert that the function returns an error or an appropriate message indicating that the input is invalid.

Now turning to valid inputs. The input domain can be divided into several equivalence classes (EC):

1. The file name is an empty string.
2. The file name is a non-empty string.
3. The file extension has different cases.
4. The search target is an empty string.
5. The search target is a non-empty string.
6. The search target contains special characters.
7. The search target is not in the file.
8. The search target contains multiple products.

The coverage criteria are:

- For each EC, we need to test at least one input.

This leads to the following test requirements:

R1. If the input is in EC1, the function should raise a `FileNotFoundError`.
R2. If the input is in EC2, the function should display the table content, regardless of whether the non-empty string contains special characters, spaces, etc.
R3. If the input is in EC3, the function should display the table content.
R4. If the input is in EC4, the function should display only the header.
R5. If the input is in EC5, the function should display the table content, regardless of the case of the non-empty string.
R6. If the input is in EC6 - 7, the function should display only the header.
R7. If the input is in EC8, the function should display multiple target products, regardless of whether it contains special characters.

This leads to the corresponding test cases:

TC1. Copy the original file and name it ""
TC2. Copy the original file and name it "products_copy.csv", "products copy.csv", "prod%ucts@copy.csv" or "PROdUCTS.csv"
TC3. Copy the original file and name it "products.CSV"
TC4. The search target is ""
TC5. The search target is "apple", "APPLE", "aPpLe"

TC6. The search target is "aPp@e"
TC7. The search target is "house"
TC8. The search target is "apple, banana", "apple&banana"

Unit Testing of `search_and_purchase_product`

The `search_and_purchase_product` function accepts user input to search for products and guides the purchase process. Invalid inputs include non-string types for the product search query. I create one test for each of these types and assert that the function appropriately handles the error.

Now turning to valid inputs. The input domain can be divided into several equivalence classes (EC):

1. The user inputs a product name.
2. The user inputs multiple product names.
3. The user inputs "all".
4. The user inputs "all" and product names.

The coverage criteria are:

- For each EC, we need to test at least one input.

This leads to the following test requirements:
R1. If the input is in EC1, the `display_filtered_table` function should be called once, regardless of whether the product name is valid or not.
R2. If the input is in EC2, the `display_filtered_table` function should be called multiple times.
R3. If the input is in EC3, the `display_csv_as_table` function should be called the corresponding number of times.
R4. If the input is in EC4, the `display_filtered_table` and `display_csv_as_table` functions should be called the corresponding number of times according to the input.

This leads to the corresponding test cases:
TC1. Input = ['HOUSE', 'y'], ['Apple', 'y'], ['apPle', 'Y']
TC2. Input = ['Apple', 'n', 'Banana', 'y']
TC3. Input = ['all', 'y'], ['AlL', 'Y'], ['all', 'n', 'all', 'y']
TC4. Input = ['all', 'n', 'Apple', 'y'], ['all', 'n', 'Apple', 'n', 'Banana', 'y'], ['LAPTOP', 'n', 'all', 'y'], ['Apple', 'n', 'Grapes', 'n', 'all', 'y'], ['all', 'n', 'Apple', 'n', 'Banana', 'n', 'all', 'y']