Test Report for Client Server solution of file management.

The following unit tests are conducted on the class User:

What and how it is tested:

1. Repr

The simplest unittest is testing User class repr, we wrote this test to make sure we had set up the tests properly, and that the testfile could create new User objects.

2. User Registration & Login:

Testing conducted to test whether users can be registered successfully when they give input such as username, password and privileges. If they try to register with same username one more time they receive the message "Username already exist. Please enter valid username". Due to some restrictions in return statement from Class User file, we have verified this testcase by slicing particular part of output strings which displays "User registered sucessfully and User already exists. At the end of the test, the registered user is deleted.

3. Change directory:

Testing conducted to check whether the user can change folder in the current directory by giving name of the folder and if they give input as (..) they can move folder backwards but this should only happen if the user is not in root. In case if the user try to give folder name that does not exists they will get error message in this scenario. We test this by moving to the root folder, then moving to the folder Users, and comparing the return from the function to confirm.

4. Write file:

Testing conducted to check whether the user is able to create file with text, the result we get for running this script is "file is created" in the second time if we run this script with same file name, the file is getting updated with user input. The message printed should be "File is updated". If we pass the empty input, the content of the file is erased. After running this testscript, we are deleting the file so as to not get any error when we run this script next time

5. Create directory:

Testing conducted to create folder in the current working directory. User can able to see the folder in the specified location after this test run. But due to frequent use of testscripts we include a command to delete the folder after printing results. Otherwise when we run this test case next time it would fail. If we create a directory that already exists it throws message such as directory name already exists. After running this test, we are deleting the folder to clean up

after the test.

6. List function:

Testing conducted to print the file and directory list in the current working directory. Here we have checked the result of this testcase showing the result of trying to list the contents of an empty folder .

7. Read with no input

This tests the functionality of sending the read file command without any file name specified. It should return the output that the previously read file has been closed. We test this by assignming a dummy name to the user objects self.filename, and then call the function to remove it, and see if we get the correct response.

8. Write with no input

This tests the functionality of sending the write command with a filename, but no input text. Instead of just appending an empty string (which would have no effect), it should erase the content of the file. We test this by sending the command and confirmin we get the correct message back.

9. Combine create folder and write file:

Testing conducted to combine two scenarios such as create folder and write file to test whether the functionality is working fine if we run it in same test. We confrim we receive the correct return strings from these functions, including the names of file and folder. After running this test, we are deleting the folder and file to not get an error when we run this script next time.

10. Read and write file:

Testing conducted to combine both reading and writing file. First we have to create file which contains text using writefile function and then read the same file by calling readfile function, to check it is possible to do both cases together. The test checks that the output from writefile is correct, and that the output from readfile is correct, including the string we wrote to the file using writefile. After running this testscript, we are deleting the file in order to not get an error when we run this script next time.

11. Reading an empty file

Here we try to open and read an empty file, and confirm that the read function can handle this scenario and returns the correct response, without crashing.

12. Change directory outwards

Here we test that the change folder function does not allow a user to leave their home folder by sending (..), which we confirm by setting the user object's current path attribute to their

home folder, and trying to leave, which should produce the correct error message.

Reason for doing these testcases:

To Verify the behaviour of functionality working independently from other parts and also isolated from the environment.To check the functionality of code works fine if we change input variable as expected.

Some scenarios not covered in this unittest implementation are as follows:

1. List function with a folder containing files and directory because results are complicated to confirm with assert since the string would include a lot of tabs.

2. In Reading file, basic functionality is done in unittest. We don't include a test for cycling through a very large file.

3. Logging out and delete are not included in unittest.

4. More combinations and negative test cases. We have some combined testcases like read and write, as well as create a new folder and write file, but more complex testcases could be created

5. Async functions:

A problem we were not able to solve directly was using the unittest module together with asyncio. In order to test the client-server connection with unittest, both the client and the server would have to be started by the test script. We tried to find a solution for example by using threading, or async-functions, but unittest does not accept coroutines. By searching for how to solve this, we know it can be done with some wrapper functions and threading, or by making changes to unittest, but we decided instead to not use unittest for the parts of out code which involves async. However, we could use unittest for a large part of our code: the functions on the server, both the general ones (login and register) and the user specific ones (such as list, read_file, write_file etc.).

Since we could not test the actual client-server connection with unittest, this is were we decided to use assert statements. We have assert tests in both the client and server files, which check that the other one's IP-address and TCP-port have reasonable values, and which confirm that StreamReader and StreamWriter have been assigned properly. If there is a problem with the connection, it can be caught by these assert statements.

The number of tests

We have 12 tests implemented using the unittest module, which we think is enough although we do not test every function or every combination. We test a variety of functions both general ones on the server and specific to the user. Server-client connection is only tested with asserts, but our implementation is based closely on standard implementation of client-server with async and StreamReader/StreamWriter, which we read about in the Python online reference.