```python
import pandas as pd

class BookLover():
    def __init__(
        self, name, email, fav_genre,
        num_books = 0,
        book_list = pd.DataFrame({’book_name’:[], ’book_rating’:[]})):
        """
        Initializes the BookLover with the given name, email, favorite genre, number of books read, and book list.


        INPUTS:
        name : str
            The name of the book lover.
        email : str
            The email of the book lover.
        fav_genre : str
            The favorite genre of the book lover.
        num_books : int, optional, default = 0
            The number of books the book lover has read.
        book_list : pandas.DataFrame, optional
            A DataFrame containing the names and ratings of the books read (default is an empty DataFrame).
        """
        self.name = name
        self.email = email
        self.fav_genre = fav_genre
        if num_books != book_list.shape[0]:
            self.num_books = book_list.shape[0]
        else:
            self.num_books = num_books
        if book_list.shape[0] == book_list.drop_duplicates([’book_name’]).shape[0]:
            self.book_list = book_list
        else:
            raise ValueError("book_list must contain unique books")

    def add_book(self, book_name, book_rating):
        """
        Adds a book to the book list if it hasn’t been read yet.

        INPUTS:
        book_name : str
            The name of the book to add.
        book_rating : int
            The rating of the book to add.
        """
        if self.has_read(book_name) == False:
            new_book = pd.DataFrame({
                ’book_name’: [book_name],
                ’book_rating’: [book_rating]
            })

            self.book_list = pd.concat([self.book_list, new_book], ignore_index=True)
            self.num_books += 1


    def has_read(self, book_name):
        """
        Check if book has already been read.

        INPUT:
        book_name : str
            The name of the book to check.

        OUTPUT:
        bool
            True if book has been read, else False.
        """
        if book_name in self.book_list['book_name'].values:
            print(f"{self.name}, This Book, '{book_name}' Already Exists in Your Book List")
            return True
        else:
            return False

    def num_books_read(self):
        """
        Returns the number of books read.

        OUTPUT:
        int
            The number of books read.
        """
        return self.num_books

    def fav_books(self):
        """
        Returns a DataFrame of books with a rating greater than 3.

        OUTPUT:
        pandas.DataFrame
            A DataFrame with the favorite books.
        """
        return self.book_list.loc[self.book_list['book_rating']>3,:]import unittest
import numpy as np
import pandas as pd
from booklover import BookLover
import booklover as bl

class BookLoverTestSuite(unittest.TestCase):

    def test_1_add_book(self):
        """
        add a book and test if it is in `book_list`.
        """
        book = BookLover("Name","Email","NA")
        book.add_book("Travels with Charley",5)
        self.assertTrue("Travels with Charley" in book.book_list['book_name'].values)

    def test_2_add_book(self):
        """
        add the same book twice. Test if it's in `book_list` only once.
        """
        book = BookLover("Dude","Email","NA")
        book.add_book("Travels with Charley",5)
        book.add_book("Travels with Charley",5)
        self.assertEqual(book.book_list[book.book_list['book_name'] == "Travels with Charley"].shape[0],1)

    def test_3_has_read(self):
        """
        pass a book in the list and test if the answer is `True`.
        """
        book = BookLover(
         "Dude","Email","NA",1,
         pd.DataFrame({'book_name':["Travels with Charley"], 'book_rating':[5]}))

        self.assertTrue(book.has_read("Travels with Charley"))

    def test_4_has_read(self):
        """
        pass a book NOT in the list and use `assert False` to test the answer is `True`
        """
        book = BookLover(
         "Dude","Email","NA",1,
         pd.DataFrame({'book_name':["Travels with Charley"], 'book_rating':[5]}))

        self.assertFalse(book.has_read("Travels without Charley"))

    def test_5_num_books_read(self):
        """
        add some books to the list, and test num_books matches expected.
        """
        book = BookLover("Dude","Email","NA")
        book.add_book("Traveled with Charley",5)
        book.add_book("Travels with Charley",5)
        book.add_book("Travels with Charley",5) #DUP
        book.add_book("Traveling with Charley",5)
        book.add_book("Travel with Charley",5)
        book.add_book("Will Travel with Charley",5)
        self.assertEqual(book.num_books,5)


    def test_6_fav_books(self):
        """
        add some books with ratings to the list, making sure some of them have rating > 3.Â

        Your test should check that the returned books have rating  > 3
        """
        book = BookLover("Dude","Email","NA")
        book.add_book("Traveled with Charley",1)
        book.add_book("Travels with Charley",4)
        book.add_book("Travels with Charley",5) #DUP
        book.add_book("Traveling with Charley",3)
        book.add_book("Travel with Charley",2)
        book.add_book("Will Travel with Charley",5)
        self.assertTrue(all(book.fav_books()['book_rating'].values > 3))

if __name__ == '__main__':
    unittest.main(verbosity=3)test_1_add_book (__main__.BookLoverTestSuite.test_1_add_book) ... ok
test_2_add_book (__main__.BookLoverTestSuite.test_2_add_book) ... ok
test_3_has_read (__main__.BookLoverTestSuite.test_3_has_read) ... ok
test_4_has_read (__main__.BookLoverTestSuite.test_4_has_read) ... ok
test_5_num_books_read (__main__.BookLoverTestSuite.test_5_num_books_read) ... ok
test_6_fav_books (__main__.BookLoverTestSuite.test_6_fav_books) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.010s

OK
```