Testing

Testing for development

```python
def arrange_scores():
    # Open file in r+ mode and read all lines
    with open("testfile.txt", "r+") as Score_Manager:
        lines = Score_Manager.readlines()
        # Extract scores
        lined = lines[3:]
        # Extract header
        header = lines[:3]
        # Sort scores
        lined.sort()
```

The problem how to sort the score in descending order not ascending

```python
# Function to sort scores in descending order
def arrange_scores():
    # Open file in r+ mode and read all lines
    with open("testfile.txt", "r+") as Score_Manager:
        lines = Score_Manager.readlines()
        # Extract scores
        lined = lines[3:]
        # Extract header
        header = lines[:3]
        # Sort scores
        lined.sort(key=lambda line: int(line.split('|')
[1].strip()), reverse=True)

    # Open file in w mode and rewrite the sorted scores
    with open("testfile.txt", "w") as Score_Manager:
        # Write header
        Score_Manager.writelines(header)
        # Write sorted scores
        Score_Manager.writelines(lined)
```

I solved by using function key=lambda and reserve the ascending order to descending

This function opens a file in read and write mode, reads all the lines, sorts the scores in descending order with extracts the header.

```python
def score(new_name, new_score):
    # Open file in a+ mode
    with open("testfile.txt", "a+") as Score_Manager:
        # Check if 'new_score' is less than 0; if not, proceed to the 'else' statement.
        if new_score < 0:
            print("Score starts from 0")
            return
        else:
            # Open file using 'a+' mode and read its contents
            with open("testfile.txt", "a+") as Score_Manager:
                Score_Manager.seek(0)
                contents = Score_Manager.read()

                # Display a message (header) if the file does not contain any contents
                if not contents:
                    Score_Manager.write("High scores manager\n".center(75))
                    Score_Manager.write("\n player" + "|".center(24) + "Top scores\n".center(10))
```

```python
# Separate file contents into lines by newline (\n), removing spaces, and store as lines
lines = [line.strip() for line in contents.split('\n')]
# Loop to check every line; split the line by |
for line in lines:
    parts = line.split('|')

    # Assigned variable containing the name of the player only
    name = parts[0].strip()
    # Check if the new name exists in the file. If so, display a message and return
    if new_name == name:
        print("The name exists")
        return
# Write new_name and new_score to the file
Score_Manager.write(f"{new_name:<17} | {new_score:^30}\n")
```

The problem when user enter name and enter same name but with space considered different which needs to considered them same name

```python
# Separate file contents into lines by newline (\n), removing spaces, and store as lines
lines = [line.strip() for line in contents.split('\n')]

# Loop to check every line; split the line by |
for line in lines:
  parts = line.split('|')

  # Assigned variable containing the name of the player only
  name = parts[0].strip()

  # Remove extra spaces only from the beginning of new_name for comparison

  new_name_stripped = new_name.lstrip()

  # Check if the stripped names are equal
  if new_name_stripped == name:
      print("The name exists")
      return


# Write new_name and new_score to the file
Score_Manager.write(f"{new_name:<17} | {new_score:^30}\n")
return
```

Solved by use the lstrip() function in my Python code to trim any extra space from the beginning of the first name only. but, there might be extra space in the second name, so considered different name.

```python
def score(new_name, new_score):
    # Open file in a+ mode
    with open("testfile.txt", "a+") as Score_Manager:
        # Check if 'new_score' is less than 0; if not, proceed to the 'else' statement.
        if new_score < 0:
            print("Score starts from 0")
            return
        else:
            # Open file using 'a+' mode and read its contents
            with open("testfile.txt", "a+") as Score_Manager:
                Score_Manager.seek(0)
                contents = Score_Manager.read()

                # Display a message (header) if the file does not contain any contents
                if not contents:
                    Score_Manager.write("High scores manager\n".center(75))
                    Score_Manager.write("\n player" + "|".center(24) + "Top scores\n".center(10))
            # Separate file contents into lines by newline (\n), removing spaces, and store as lines
            lines = [line.strip() for line in contents.split('\n')]

            # Loop to check every line; split the line by |
            for line in lines:
              parts = line.split('|')

              # Assigned variable containing the name of the player only
              name = parts[0].strip()

              # Remove extra spaces only from the beginning of new_name for comparison

              new_name_stripped = new_name.lstrip()

              # Check if the stripped names are equal
              if new_name_stripped == name:
                  print("The name exists")
                  return


            # Write new_name and new_score to the file
            Score_Manager.write(f"{new_name:<17} | {new_score:^30}\n")
            return
```

The function score(new_name, new_score) ensures appending mode rejects invalid entries, adds a header if file is empty, checks if new_name exists, and writes new_name, new_score only if conditions didn't met, else end the function when any of conditions met.

```python
print("Hello our user to high score manager")
print("Please provide your name and score\n")
```

The problem: When attempting to search for or update a user or file that doesn't exist, errors occur.

```python
# Display messages and options to the user
print("Hello our user to high score manager")
print("Please choose your preferred option\n")
# Create and terminate a file to establish its existence
with open("testfile.txt", "a+") as Score_Manager:
 Score_Manager.close()
```

I addressed the issue by ensuring that the file exists before performing any operations.

```python
while True:
    # Display options for user input
    option = input("\n(add scores |1| || search  |2|  update |3| || exit |4|) : ")

    # Check if the user chooses option 1
    if option == "1":
        # Request the user to input their name and score
        name = input("\nEnter your name: ")
        user_score = input("Enter your score: ")
```

```python
        try:
            # Try if name input or user_score input empty, display a message
            if name == "" or user_score == "":
                print("Empty inputs are invalid")
            # Else call score function and arrange_scores function
            else:
                score(name, int(user_score))
                arrange_scores()
        # Except if user_score is not an integer, display a message
        except ValueError:
            print("Score with numbers only")
# return to the top of the loop
```

The program starts by confirming file existence, then enters a loop for user options; if '1' is chosen, it prompts for name and score, writes them to the file under conditions, and sorts by scores; an exception handles non-integer scores, returning to the option list.

```python
    elif option == "2":
        # Display message then request the user to input search_name
        print("\nSearching for scores.....\n")
        search_name = input("Enter the player's name without unnecessary spaces : ")
        # Assigned found variable to false
        found = False
```

```python
        # Open file in r+ mode
        with open("testfile.txt", "r+") as Score_Manager:
            # Loop to check every line; split the line by |
            for line in Score_Manager:
                parts = line.split('|')
                # Check if search_name equals parts[0].strip()
                if search_name == parts[0].strip():
                    # Display the line related and set found to true
                    print(line)

        # Executed if the previous condition fails or if the search_name input is empty, displaying a message.
        if search_name == "":
            print("No matching names found\n")
# return to the top of the loop
```

```python
        # Open file in r+ mode
        with open("testfile.txt", "r+") as Score_Manager:
            # Loop to check every line; split the line by |
            for line in Score_Manager:
                parts = line.split('|')
                # Check if search_name equals parts[0].strip()
                if search_name == parts[0].strip():
                    # Display the line related and set found to true
                    print(line)
                    found = True
        # Executed if the previous condition fails or if the search_name input is empty, displaying a message.
        if not found or search_name == "":
            print("No matching names found\n")
# return to the top of the loop
```

The problem : If the name isn't found in the file, the program needs to manage this situation.

I resolved it using Boolean values. To prevent issues, I integrated it into the search function when name found to avoid the message once the name is found.

Prompt the user to input a name to search for, then read a file, checking for matches with the inputted name, and display the corresponding line if found, or notify the user if not and return to the options then return to option list.

```python
elif option == "3":
    # Display message then request the user to input update_name and input new_score1
    print("\nUpdating scores.....")
    update_name = input("\nInput player name for score update without extra spaces : ")
    new_score1 = input("Enter the new score: ")
    # check if update_name input or new_score1 input empty, display a message
    if new_score1 == "" or update_name == "":
        print("Empty inputs are invalid")

      # excute when previous statments fail
    else:
        # score_exists1" is set to False, while scores and updated lines assigned as lists
        scored = []
        updated_lines = []
        score_exists1 = False
```

```python
elif option == "3":
    # Display message then request the user to input update_name and input new_score1
    print("\nUpdating scores.....")
    update_name = input("\nInput player name for score update without extra spaces : ")
    new_score1 = input("Enter the new score: ")
    # check if update_name input or new_score1 input empty, display a message
    if new_score1 == "" or update_name == "":
        print("Empty inputs are invalid")
    # check elif new_score1 is alphabet or not positive numbers ,display a message
    elif new_score1.isalpha() or not new_score1.isnumeric():
        print("Score with postive numbers only")
      # excute when previous statments fail
    else:
        # score_exists1" is set to False, while scores and updated lines assigned as lists
        scored = []
        updated_lines = []
        score_exists1 = False
```

The problem: The issue lies in the update process where scores must strictly be positive numbers, not any other type of input.

write condition which if you type anything other than positive integers, it displays a message and returns you to the options. Otherwise, it proceeds with a different action.

```python
            # Open file in r+ mode and read all lines
            with open("testfile.txt", "r") as Score_Manager:
                lines = Score_Manager.readlines()
            # Loop to check every line; split the line by |
            for line in lines:
                parts = line.split('|')
                #  Check if update_name equals parts[0].strip()
                if parts[0].strip() == update_name:
                  #  Set score_exists1 to True and replace parts[1].strip() with new_score1
                    score_exists1 = True
                    updated_line = line.replace(f" {parts[1].strip()} ", f" {new_score1} ")
                    # Append updated_line to updated_lines and display messgae
                    updated_lines.append(updated_line)
                    print("Score successfully updated")
                  # excute when previous statments fail
                else:
                    # appened the empty list scored to line
                    scored.append(line)
            # If "score_exists1" is still false, display the message
            if not score_exists1:
                print("Name not found")
            # else Open file in "w" mode, add updated lines with scored to replace old scores
            else:
              with open("testfile.txt", "w") as Score_Manager:
                  Score_Manager.writelines(scored + updated_lines)
              # call the function to sort scores in descending order
              arrange_scores()
# return to the top of the loop
```

Option 3 lets users input player name and score for updating. It checks for valid inputs, updates scores if the name matches, or displays a message if not found. Finally, it sorts scores and returns the option list.

```python
elif option == "4":
    #display message
    print("\nThank you for using the high score manager ")
    print("\nNote, the file saved any changes")
```

```python
# Execute when beyond the range of available options
else:
    # display message then return to the top of the loop
    print("\nChoose from available options. Try again")
```

```python
# Check elif the user chooses option 4
elif option == "4":
    #display message then exit from program
    print("\nThank you for using the high score manager ")
    print("\nNote, the file saved any changes")
    exit()
# Execute when beyond the range of available options
else:
    # display message then return to the top of the loop
    print("\nChoose from available options. Try again")
```

The problem: When the user inputs the number 4, display the message indicating the end of the program, and then return to the list of options.

The issue was resolved by using the exit() function, which effectively terminates the program

If Option 4 is selected, show a message to the user and end the program. Otherwise, if the provided option number doesn't correspond to any available option, show a message and provide the list of available options.

Testing for Evaluation

	Input testing		results	Test purpose
option	1		Move the loop to option 1	Display the inputs inside the option
	2		Move the loop to option 2	Display the inputs inside the option
	3		Move the loop to option 3	Display the inputs inside the option
	4		Move the loop to option 4	Display the message inside the option
	rest of values		Choose from available options. Try again	Display the message inside the option
name user_score	Empty Mohamed	13 Empty	Empty inputs are invalid	To check the validity of values
	Name exist New Name	Negative numbers Negative numbers	Score starts from 0	To check the validity of values
	Name exist	positive numbers	The name exists	To check the validity of values
	New Name	postive numbers	call the function	To write the scores and sort them
	Name exist or new name	non numeric	Score with numbers only	To check the validity of values
search_name	name not exist in the file		No matching names found	To handle case of not exist of name
	name exist in the file		display line	To handle case exist of name and display its score line
update_name new_score1	Empty Mohamed	13 Empty	Empty inputs are invalid	To check the validity of values
	New Name exist Name exist	Negative numbers Negative numbers	Score with postive numbers only	To check the validity of values
	Name not exist	positive numbers.	Name not found	To check the validity of values
	Name exist	positive numbers.	Score successfully updated	To update the score with write and sort it

How the final program tested ?

1-Run the program.

2-Test Cases:

Adding Scores (Option 1):

- Test input scores with valid inputs.
- Test without enter an input.
- Test input scores with non-numeric scores.

Searching Scores (Option 2):

- Test input for existing names.
- Test input for non-existing names.
- Test without enter an input.

Updating Scores (Option 3):

- Test input scores for existing names.
- Test input scores for non-existing names.
- Test input scores with valid inputs.
- Test without enter an input.
- Test input scores with non-numeric scores.

Exiting (Option 4):

- Test if the program exits gracefully.

options (out of range):

- **Test if return to the option list**

3- scale Testing :

- Test the program with values, like very large scores or very long names.

.

4- Error Handling:

- Test error cases to confirm that error messages are displayed.

5- checking file content:

- While check the messages check that file work such as the header does not duplicate and sort scores in all operations