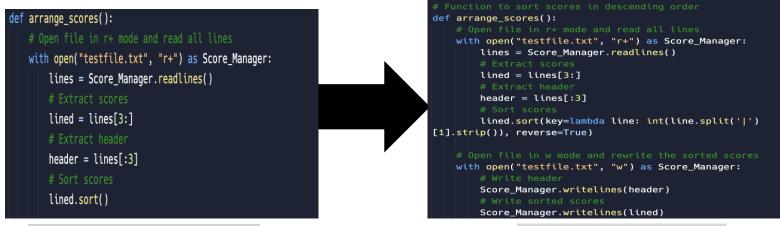
Testing

Testing for development



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

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.



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

Solved by use the Istrip() 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.

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")

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

Loss 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(#].strip()

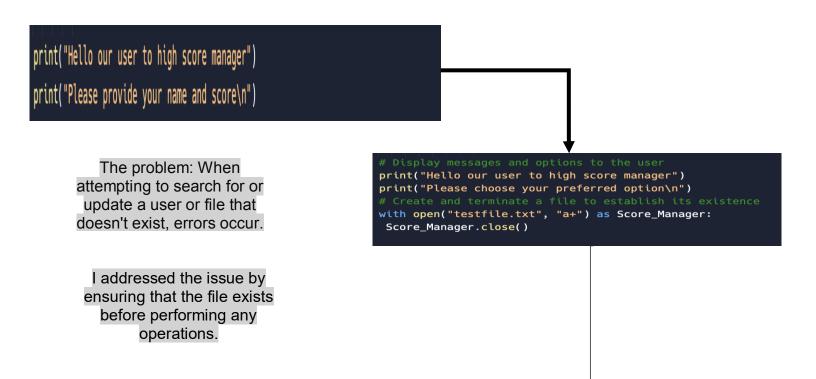
Remove extra spaces only from the beginning of new_name for comparison
new_name_stripped = new_name.lstrip()

Check if the stripped names are equal
if new_name.stripped = new_name:
 print('The name exists'')
 return

Nrise new_name and new_score to the file
Score_Namager.write(f*fnew_name:<!T} | {new_score:^38}\n')
return

```
score(new_name, new_score):
with open("testfile.txt", "a+") as Score_Manager:
      if new_score < 0:
           print("Score starts from 0")
           return
           # 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()
                 if not contents:
                       Score_Manager.write("High scores manager\n".center(75))
Score_Manager.write("\n player" + "|".center(24) + "Top scores\n".ce<u>nter(10))</u>
              # Separate file contents into lines by newline (\n), ren
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('|')
                 name = parts[0].strip()
                 new_name_stripped = new_name.lstrip()
                 if new_name_stripped == name:
                      print("The name exists")
              # Write new_name and new_score to the file
Score_Manager.write(f"{new_name:<17} | {new_score:^30}\n")
```

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.



```
# Display options for user input

# Display options for user input

# 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:

# 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: ")

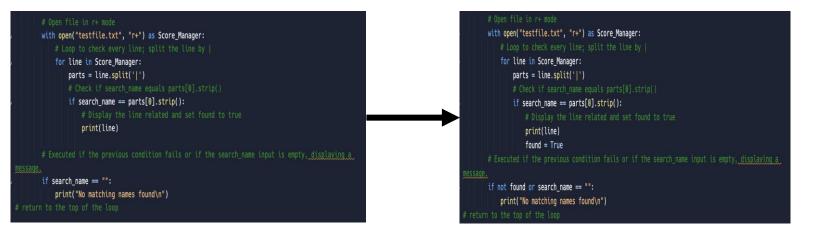
* return to the top of the loop
```

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.

```
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
```



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.

```
print("\nUpdating scores.....")
               print("\nUpdating scores.....")
                                                                                                                          update_name = input("\nInput player name for score update without extra spaces : ")
              update_name = input("\nInput player name for score update without extra spaces : ")
                                                                                                                          new score1 = input("Enter the new score: ")
              new_score1 = input("Enter the new score: ")
                                                                                                                          if new score1 == "" or update name == "":
              if new_score1 == "" or update_name == "":
                                                                                                                             print("Empty inputs are invalid")
                 print("Empty inputs are invalid")
                                                                                                                          elif new score1.isalpha() or not new score1.isnumeric():
                                                                                                                             print("Score with postive numbers only")
                  scored = []
                                                                                                                             scored = []
                  updated_lines = []
                                                                                                                             updated lines = []
                  score_exists1 = False
                                                                                                                             score_exists1 = False
         The problem: The issue lies
         in the update process where
                                                                              with open('testfile.txt", "r') as Score_Manager:
    lines = Score_Manager.readlines()
              scores must strictly be
                                                                              for line in lines:
   parts = line.split('|')
           positive numbers, not any
                 other type of input.
                                                                                  if parts[0].strip() == update_name:
                                                                                      score_exists1 = True
updated_line = line.replace(f" {parts[1].strip()} ", f" {new_score1}_")
  write condition which if you type
      anything other than positive
integers, it displays a message and
                                                                                      scored.append(line)
```

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.

with open("testfile.txt", "w") as Score_Manager:
 Score_Manager.writelines(scored + updated_lines)

if not score_exists1:
 print("Name not found")

arrange_scores()

returns you to the options.

Otherwise, it proceeds with a

different action.

else:

display message then return to the top of the loop

print("\nChoose from available options. Try again")

```
# Execute when beyond the range of available options
elif option == "4":
                                                                                  else:
    #display message
                                                                                     # display message then return to the top of the loop
    print("\nThank you for using the high score manager ")
                                                                                     print("\nChoose from available options. Try again")
    print("\nNote, the file saved any changes")
                                                               The problem: When the
                                                                                                          The issue was
                                                             user inputs the number 4,
                                                                                                        resolved by using
         elif option == "4":
                                                                 display the message
                                                                                                        the exit() function,
            #display message then exit from program
                                                               indicating the end of the
                                                                                                        which effectively
            print("\nThank you for using the high score manager ")
                                                            program, and then return to
                                                                                                          terminates the
            print("\nNote, the file saved any changes")
                                                                   the list of options.
                                                                                                             program
            exit()
                                                                   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
	1		Move the loop to option 1	Display the inputs inside the option
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	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
user score	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	Empty Mohamed	13 Empty	Empty inputs are invalid	To check the validity of values
	New Name exist Name exist	Negative numbers	numbers only	To check the validity of values
new_score1	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.

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

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