**Analysis**

REQUIREMENTS:

* Registration:
  + Name
  + Age
  + Year Group
  + UNIQUE username generated:
    - Name [first 3] + age + integer for uniqueness
  + Users stored externally
* Quiz taking:
  + Five questions
  + Variable difficulty
  + Variable answer options
    - Determined by difficulty
    - Easy: 2 options
    - Medium: 3 options
    - Hard: 4 options
  + Variable topics
  + Questions based on topic
  + Questions stored externally
* Results:
  + Results for quizzes taken generated and stored
  + Percentages of scores
  + Average, max, min, and sum of results calculated
    - WCA style averaging:
    - Remove best and worst score
    - Sum the rest and divide by number of results
  + Results sorted by topic and difficulty
  + Highest score shown between users
  + Results stored externally

POSSIBLE ADDITIONS:

* Password strength meter
* Friend system (display their scores)
* Types of question:
  + Multiple choice
  + Writing
  + Fill Blanks
* Featured quiz topic (random)
* Achievements and challenges
* Extra difficulty levels
* Graphical results shown
* Feature users:
  + Podium users (top scores)
  + Most improved user
  + Most consistent user
* Live UI menus (arrows to scroll menu options)

**Decomposition**

PROGRAM DECOMPOSITION:

* Menu:
  + Display options
    - Login
    - Create account
    - Exit
    - Info
  + Take user choice option input
  + Do action based on input
    - Call function
    - Quit
  + Close menu
* Info:
  + Display instructions on how to use and what to do
  + Close info
* Create account:
  + Ask for personal info
    - Name
    - Age
    - Year group
      * Base year group on age (choice of two)
  + Generate username based on name and age
    - First three letters of name + age
    - Make unique
      * Concatenate number to make username unique
      * Gem17 and Gem172 and Gem173 etc.
  + Ask for password input (do twice to make secure)
  + Hash password
  + Store name, age, school year, username and hashed password in file
  + Exit
* Login:
  + Retrieve user info from database file
  + Ask for username and password
  + Login if correct
    - Enter user account menu
  + Exit if incorrect
* User account menu:
  + Retrieve current user results from file
  + Display options
    - Take test
    - View user results
    - View all results
    - Log out
  + Take user choice option input
* Log out:
  + Save current user results to file
  + Return to menu
* Take test:
  + Retrieve test details from file
  + Display topic options
  + Take user topic input
  + Display difficulty options
  + Take user difficulty input
    - 1 = Easy
    - 2 = Medium
    - 3 = Hard
  + Generate questions randomly from range
    - Base on topic and difficulty
  + Display options based on difficulty
    - Easy = 2
    - Medium = 3
    - Hard = 4
  + Sort options randomly for answers
  + Take user answer input
    - Compare with answer from database
  + Repeat question function until test is over
  + Display results and score
  + Return to user account menu
* View all results:
  + Display array of scores and users
    - Sort best to worst
    - Display in a table with user, score, date, topic and difficulty
  + Exit results
* View user results:
  + Display array of scores
    - Sort by date/score (choice)
    - Display in a table with score, date, topic and difficulty
  + Exit results

FILES

* users.txt
  + Usernames
  + Passwords
* info.txt
  + Instructions
* results.txt
  + Scores
  + Topics
  + Difficulties

FUNCTIONS

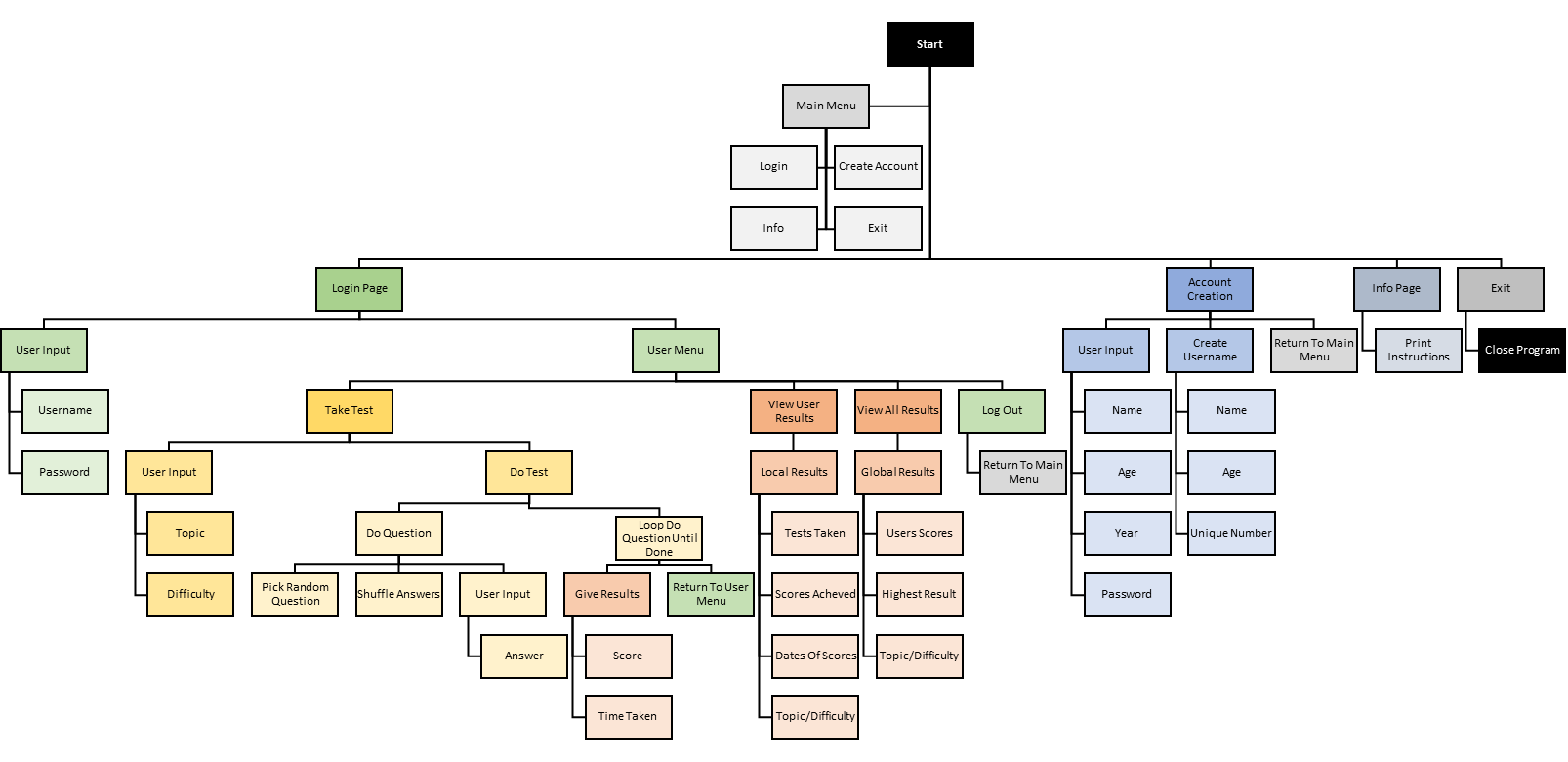
* Menu
* Create account
* Login
* User account menu
* Test question
* Test results
* User results
* All results
* Info

POSSIBLE CLASSES

* User
  + Username
  + Password
  + Scores
  + # Quizzes taken
* Quiz
  + Questions/Answers
  + Topic
  + # Questions in quiz
* Calculation
  + Type of calculation
  + Input values
  + Output values
* Menus
  + Options
  + Destinations

**Structure Diagram**

Because of the complex structure of the menu system of the program, I decided to make a structure diagram to assist the development.



**Pseudocode**

class menus:

function main\_menu:

options = list("1: Login",

"2: Create Account",

"3: Exit",

"4: Info")

display(options)

choice = input("Choose an option: ")

if choice = 1 then

go\_to\_function(login)

elseif choice = 2 then

go\_to\_function(create\_account)

elseif choice = 3 then

quit\_program()

elseif choice = 4 then

go\_to\_function(info)

endif

end function

function info:

open\_file(info.txt)

instructions = file\_contents

display(instructions)

end function

function create\_account:

name = input("What is your name? ")

age = input("How old are you? ")

possible\_yr = list(age - 4, age - 5)

yr\_group = input("Are you in year",

possible\_yr[0], "or year",

possible\_yr[1], "?")

username = name[:3] + string(age)

open\_file(users.txt)

users = file\_contents["users"]

if username in users then

username += string(count(username,users) + 1)

display("Your new username is", username, "!")

invalid = True

while invalid do

password = input("Please make a password: ")

pass\_check = input("Please enter it again: ")

if password == pass\_check then

invalid = False

endif

endwhile

hash(password)

write\_to\_file(users.txt, username, password)

endfunction

function log\_in:

username = input("Username: ")

password = input("Password: ")

hash(password)

read\_from\_file(users.txt)

if password == users\_password then

go\_to\_function(user\_account\_menu, user)

else

display("Incorrect username/password")

end if

endfunction

function user\_account\_menu:

options = list("1: Take Test",

"2: View Your Results",

"3: View All Results",

"4: Log Out")

display(options)

choice = input("Choose an option: ")

if choice == 1 then

call\_class(quiz)

elseif choice == 2 then

go\_to\_function(user\_results,user)

elseif choice == 3 then

go\_to\_function(all\_results)

elseif choice == 4 then

go\_to\_function(main\_menu)

endif

endfunction

class quiz:

function initiate:

topic = input("What topic? ")

difficulty = input("1: Easy

2: Medium

3: Hard

What difficulty? ")

score = 0

for question\_number = 0 to 5:

go\_to\_function(question,

topic,

difficulty,

question\_number)

if correct then

score += 1

endif

next question\_number

go\_to\_function(question\_results,score)

endfunction

function question(topic,difficulty,question number):

number\_questions = difficulty + 1

read\_file(quiz.exe,topic)

shuffle(questions,all\_answers)

question = questions[0]

answers = all\_answers[0][0:number\_questions]

display(question, shuffle(answers))

user\_answer = input("Which is the answer? ")

if user\_answer == answers[0] then

correct = True

else

correct = False

end if

return correct

endfunction

function question\_results(score,topic,difficulty):

display(score,topic,difficulty)

write\_to\_file(results.txt,user,score,

topic,difficulty)

endfunction

class results:

function user\_results(user):

open\_file(results.txt)

results = file\_contents[user]

display(sorted(results))

for result in sorted(results):

display((results\*100)/5)

next result

endfor

endfunction

function all\_results:

open­\_file(results.txt)

results = file\_contents

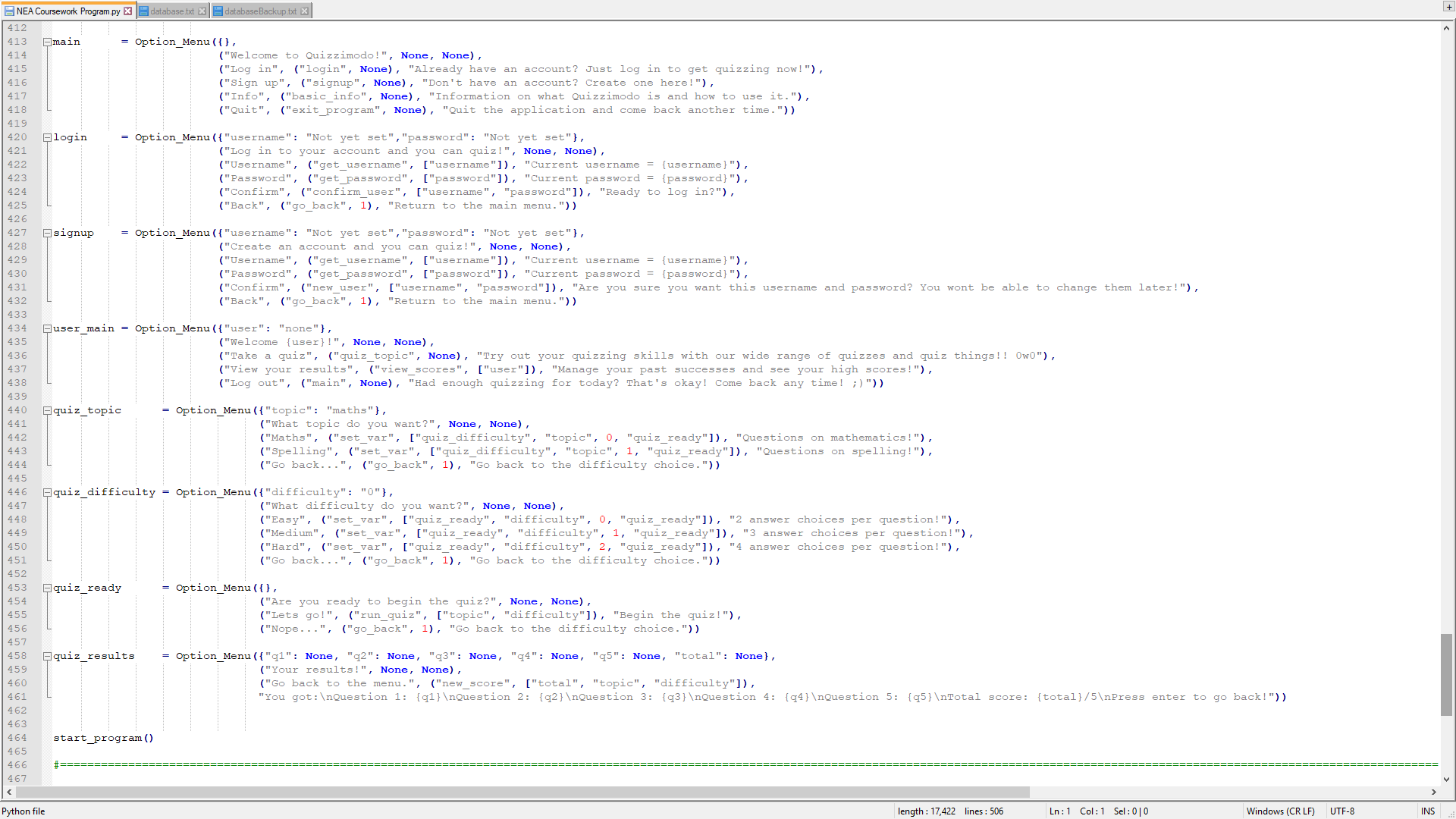
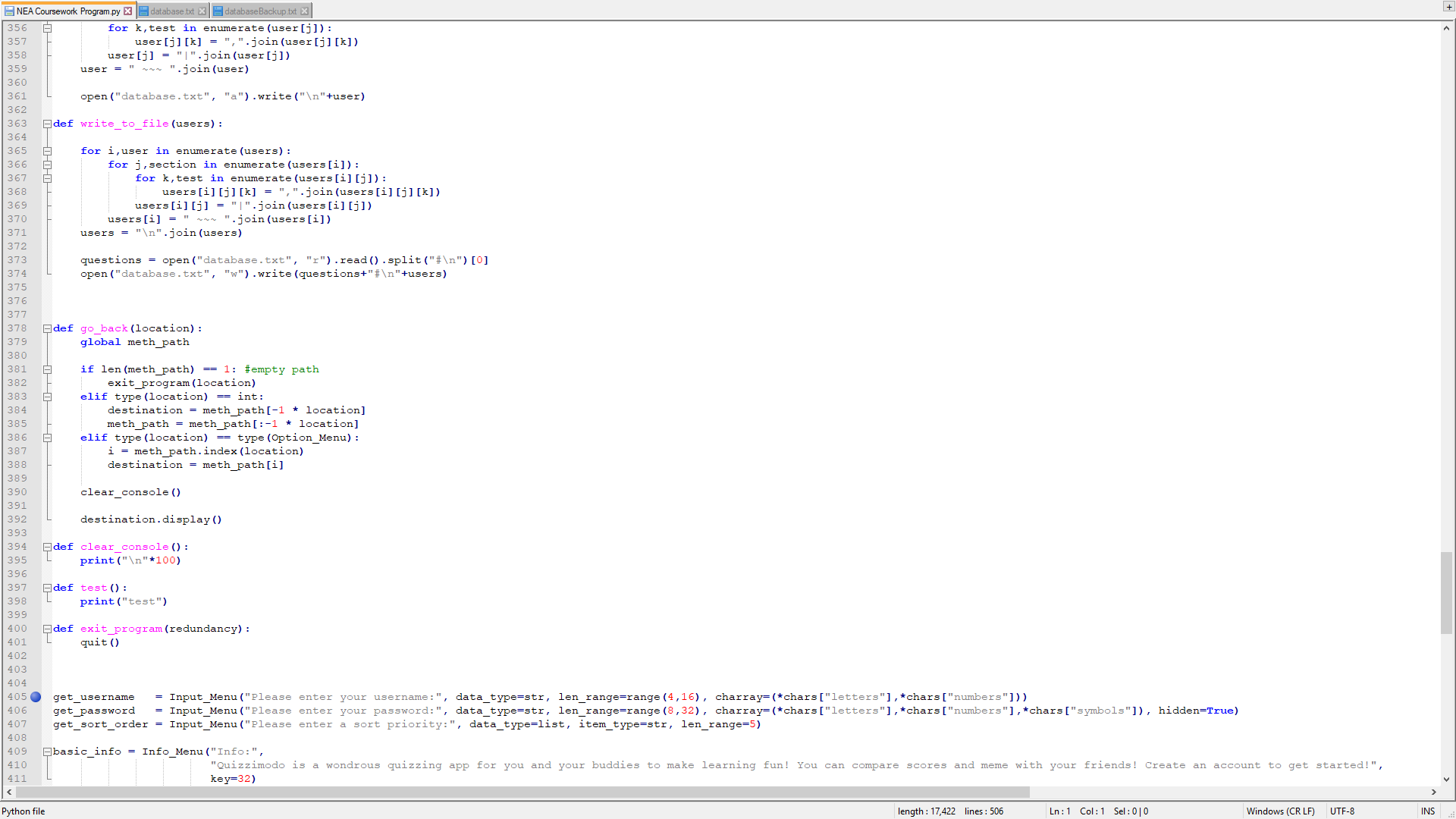
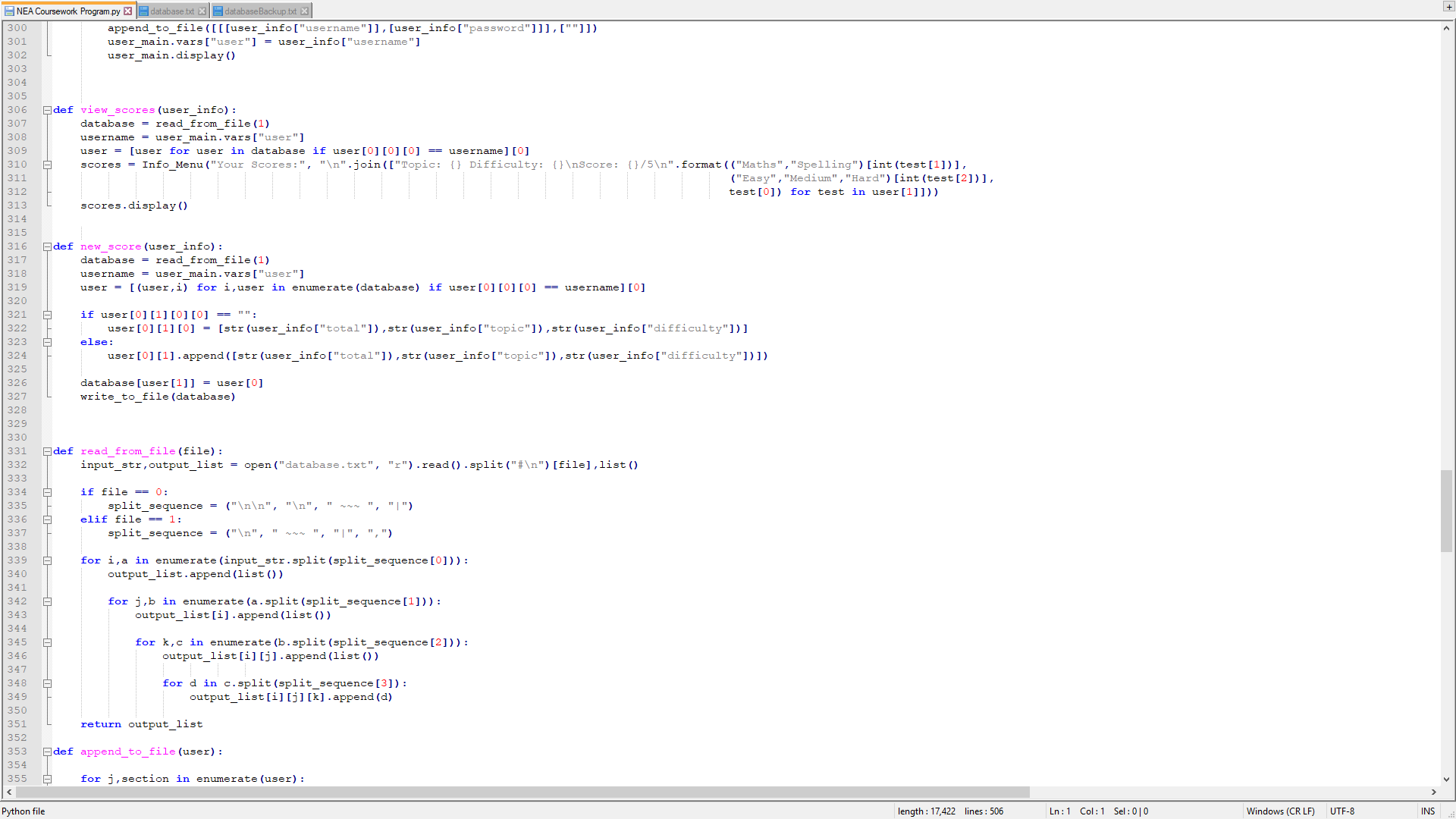
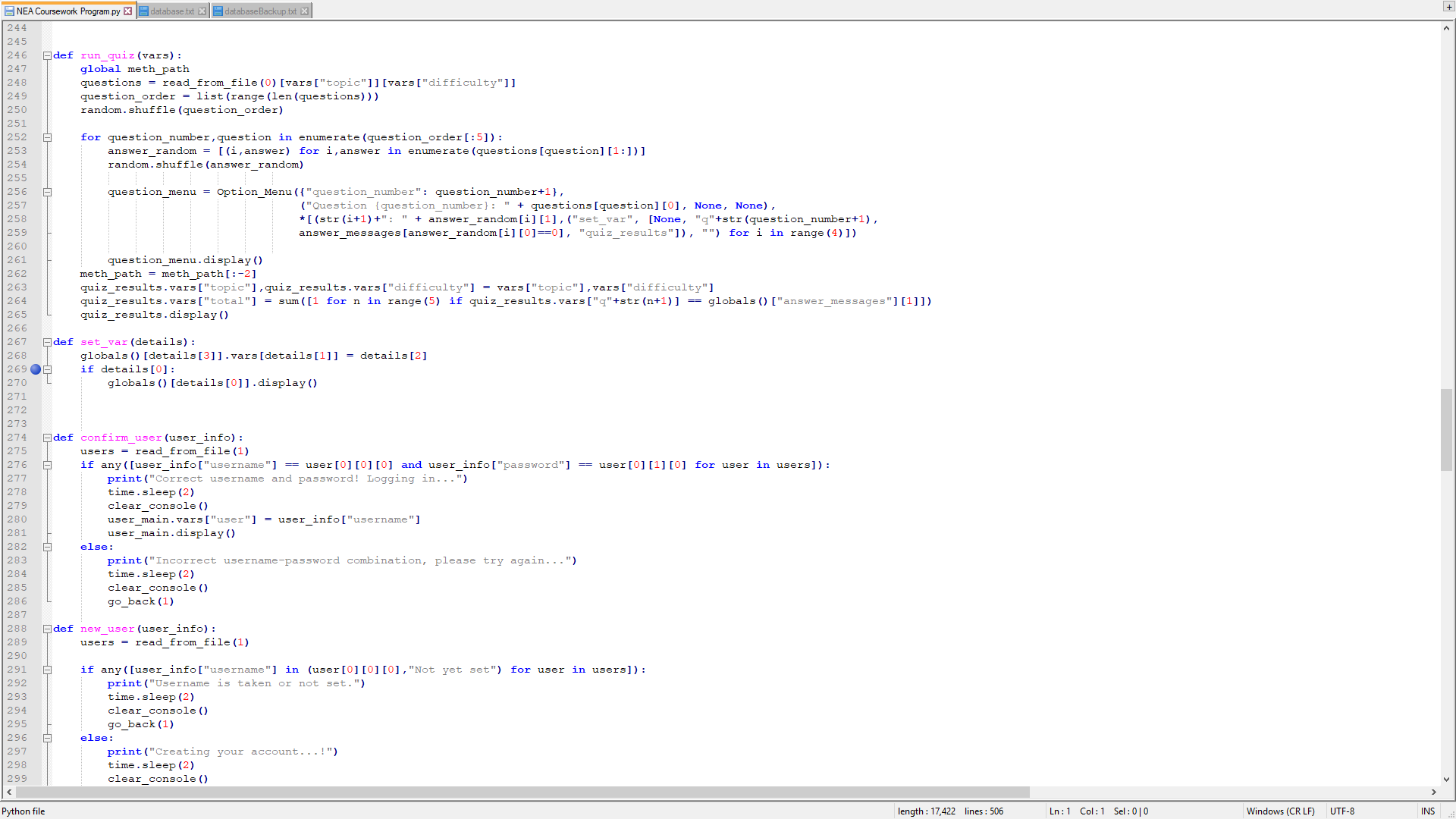
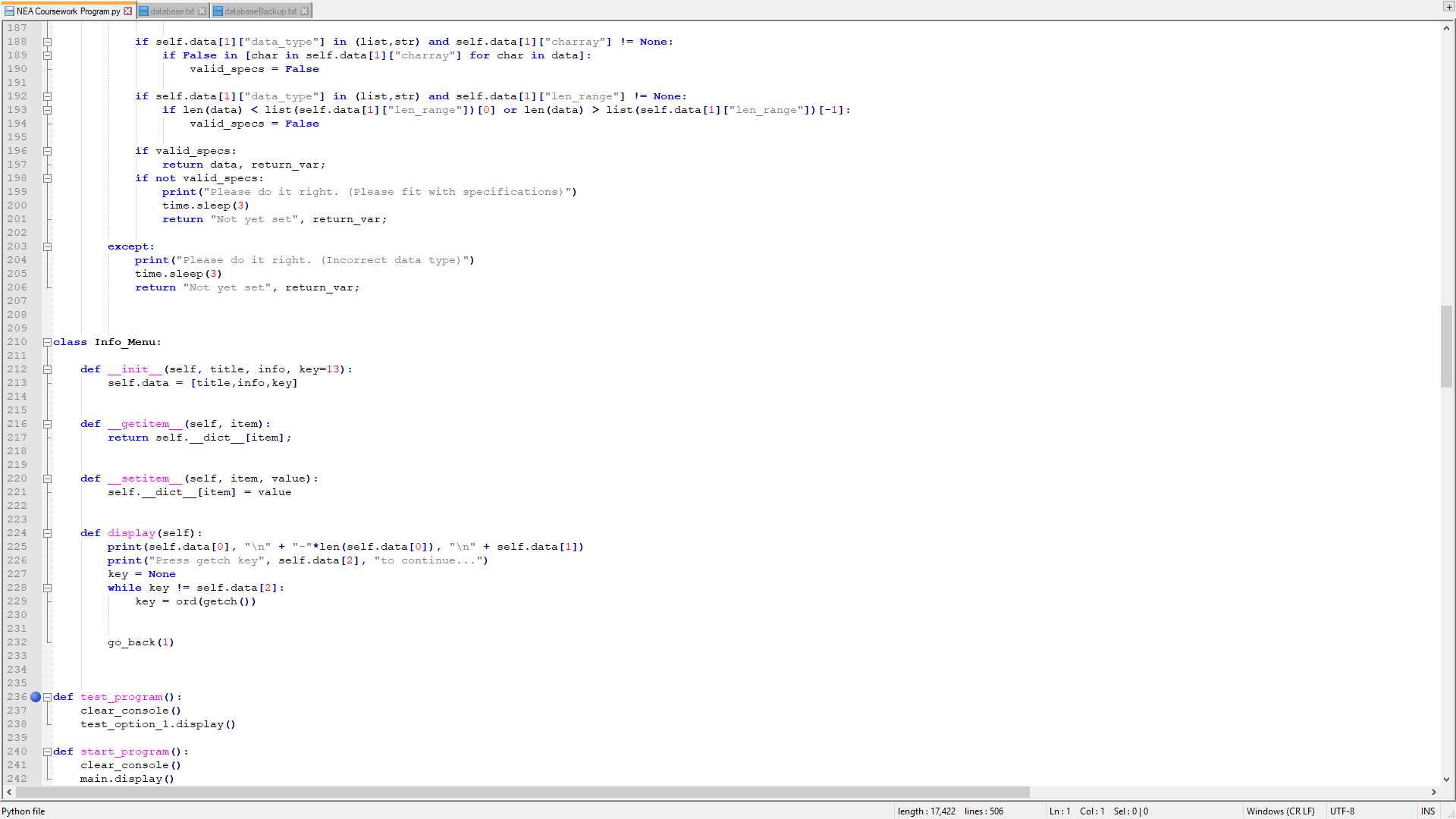
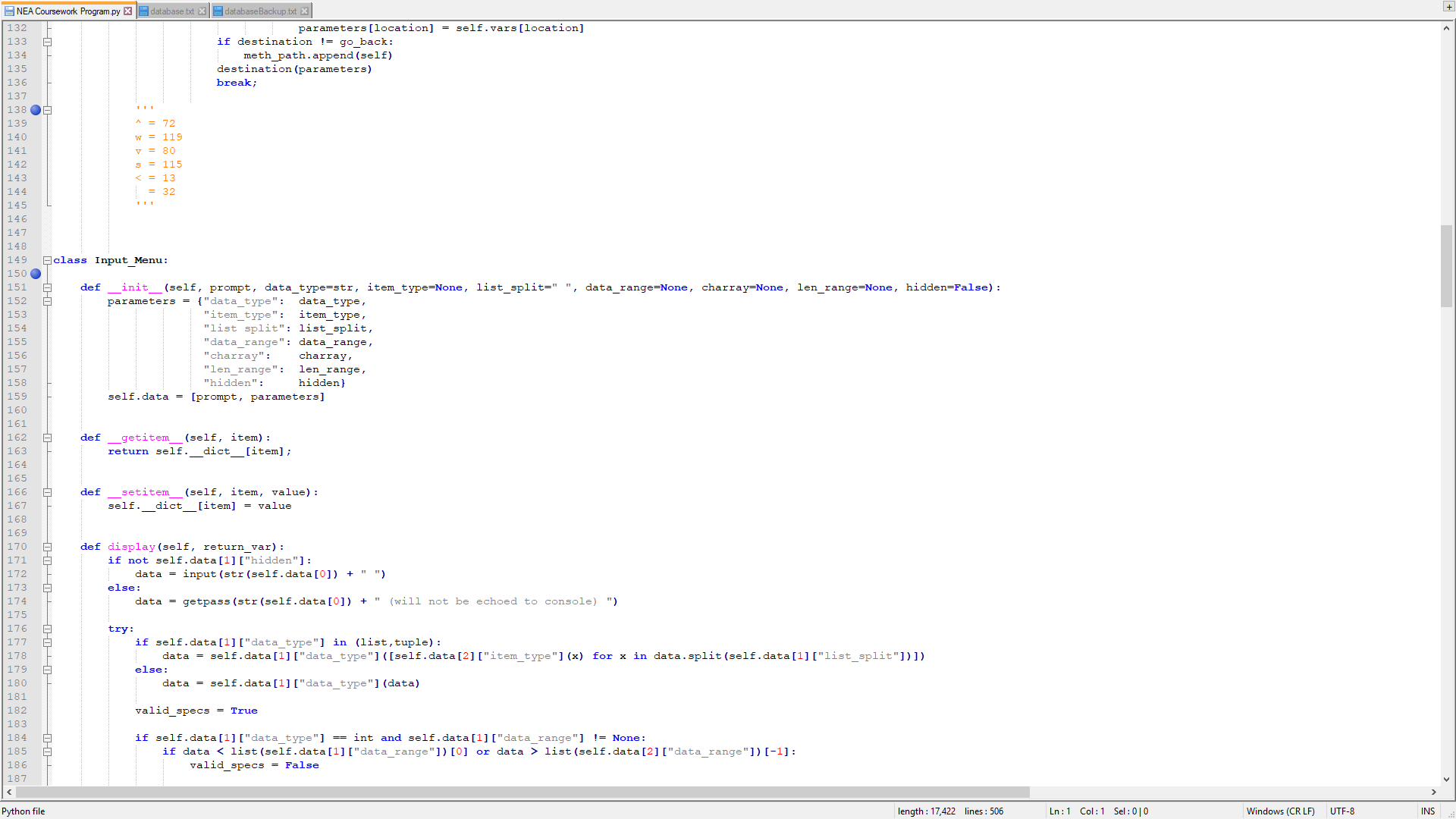
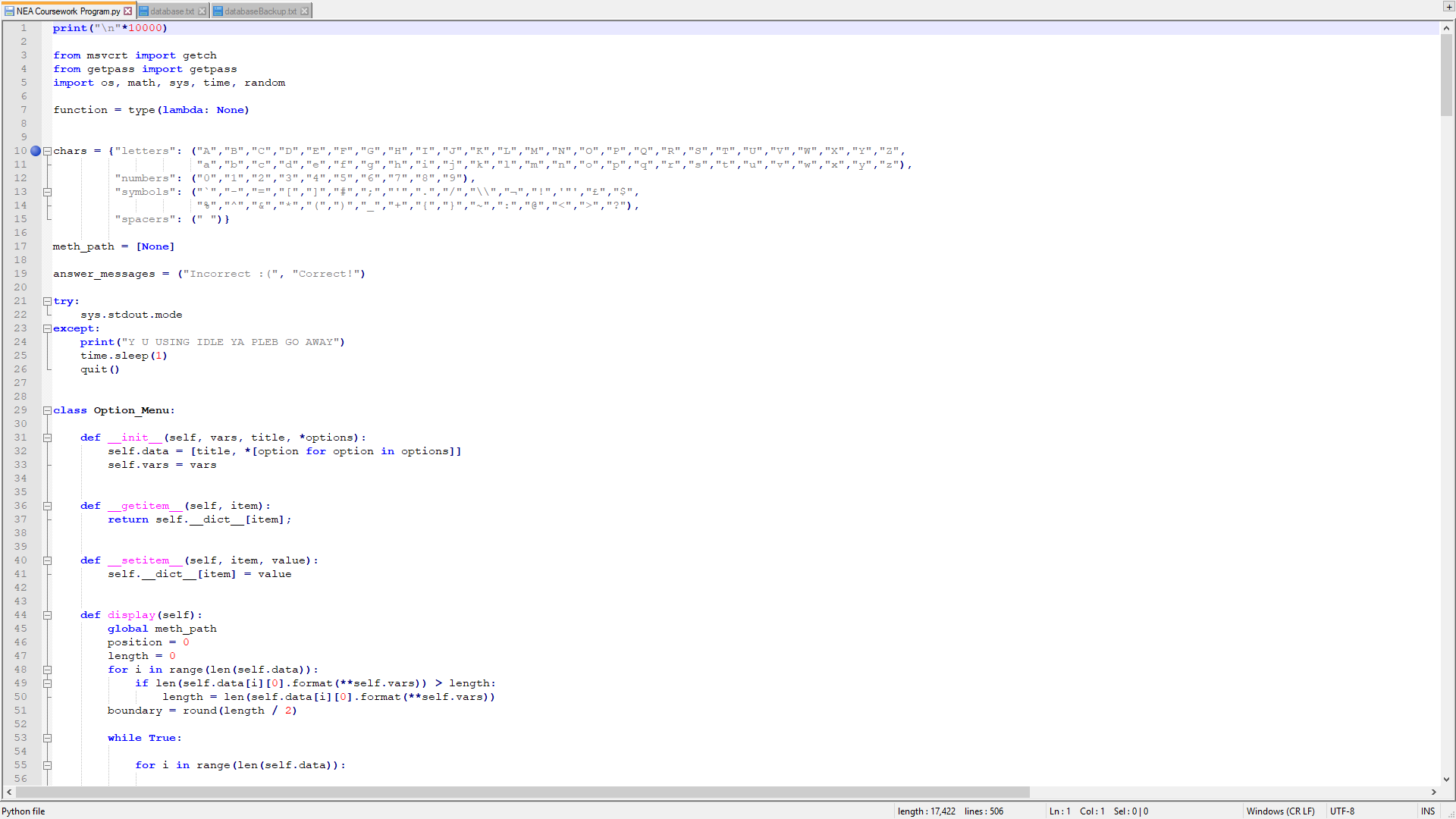
display(sorted(results))

sort = input("What do you want to sort them by?")

display(sorted(results,sort))

endfunction

**The Program**





**Documentation**

PREPARATION:

I have prepared for the development stage by producing four documents:

* Analysis:
  + Here, I have abstracted the requirements of each section of the task, such as the user registration system and the results pages. Occasionally, I have written a comment about a certain step to remind myself of a way I will do a particular thing, such as how I will structure the usernames in the system, and how I will average the results. I have also included a list of possible features I could add to the program if I have the time, to make it a better user experience. I decided to structure this document as a bullet pointed list, as I believe this makes it much easier to come back to and make use of during the development stage.
* Decomposition:
  + Here, I have broken each section and function of the program down into the step by step procedure with which I will base the development on. Each section shows what the overall structure of that part of the program will look like when completed, and includes some detail on where each function links to the rest of the program. I have also included three different lists, making note of each function, class, and file I will include in the final code. I have also included what variables each class and text file will handle.
* Structure diagram:
  + Due to the complexity of the program that I am producing, I have laid out the exact structure of the functions and processes that will be included, in a structure diagram. I have set out the four main processes that the program will involve: Account creation (blue), the login system (green), test taking (yellow), and viewing results (orange), along with some extra details on the menu system and information page. I've also added shading to show which boxes are sub-processes of others, and where return boxes will lead.
* Pseudocode:
  + Finally, I've created a 176-line pseudocode to fully plan out each function and process I've decided to include in the final program. I will mostly use the structure and design of this to design the code for the program, as it will be easy to follow this through each function I write. This pseudocode also includes a detailed process with variables and files, of how I will handle the account creation, test taking, and results viewing of this code.

DEVELOPMENT

I began by creating a simple menu function design that takes a list of a number of options and formats them into a readable, line-spaced list:

A problem that I encountered was that due to the nature of the 'for' loop and way I formatted the items out of the list, a space was added at the beginning of each non-zero value of 'i'. I dealt with this by adding a menu title to each options list, and incorperating the spaces into the design.

To do this, I added a section of code into the print line that either prints str(i) + ": ", which is the option numbering system, if i is not 0, or if i is 0, it will just print nothing, as that means it would be the title of the menu.

Next, I added in the user input for which option they want to choose. To deal with certain invalid options the user could pick, I kept the 'options' variable as a string, until I'd compared it to a list of integers I made as a global variable. If the option was an integer, I checked that it was in the range of options, and if so, returned it to the function call. My tested inputs show that any character that isn't an integer from 1 to 9, it would print "Invalid option, please choose a number: " and take the input again, and if it was not in the range of options presented, it would print "Invalid option, please chose a number within the range: " and take the input again. When I entered the option "3", the code returned 3, meaning it had taken the input.

Having make an example of how the menu would work, I checked my structure diagram and made a list for each menu I’d need with all of their options.

I tested each different menu one by one to make sure using variable menus, and having variable numbers of options would not break the program, and it ran perfectly.

When trying different option inputs, the code I'd written handled having different numbers of options well, and for the topic menu, choosing option 4 worked, but on the difficulty menu, it looped the choice, as expected.

Next, I incorporated my whole menu system into a menu class, to make it simpler to organise and create menus. I added a "Menu." to the beginning of each variable and function I would use in the menu class.

I tested the menu again, and it worked well.

Now, I implimented the different menus as class instances of the menu class, meaning I could set and get values from, and create, menus whenever I want during the code.

In order to create the user interface I’d liked to use, I had to split the class I had created into 3: Option menus, input menus, and info menus. Option menus use the arrow keys and enter key to navigate and select between a number of different options, each of which lead to either another method or a function. Input menus will return a value from the user back into the previous option menu as a variable after passing it through a set of validation to test things like string length, character sets and integer range. Info menus are much more simple, and simply display a title and body of text, along with a prompt to return to the previous option menu with a keypress.

Some functions that I need are: file reading, writing and appending, user management, score management, moving back to previous menus, clearing the console, displaying “test” and starting the program.

One problem I encountered was in the variable for the path of methods I needed to keep track of so I could move backwards to the previous method using a function. The issue was keeping track of every method that had come before, despite any odd configurations of input or info menus used. I solved this by appending the current method to a global array every time I moved into a new option menu, or into a new function except for the go\_back function. The go\_back function would then take the global list, and go to the method ‘n’ steps back from the last position on the list, while also removing n methods from the end of the array.

I decided to use object-oriented programming for my coursework because I had been learning it just before we started the project, and I believed it would make the important parts of my code reusable to base each part of the user interface on a main template I designed. This allowed the code to be much neater than if I’d just designed the code linearily, or solely with singular functions. This would have made the program much neater, however time constraints meant I couldn’t put as much detail into each individual class as I’d have liked, so I ended up resorting to functions most of the time.

I’m happy with how the program turned out, as the user interface is neat, and I have validated as much as possible without any errors in the final program. It also manages to read and write to a database file with lots of ease while keeping the file itself mostly readable. However, the code behind the interface and system is more messy than I would have liked, and could easily be cleaned up given more time to create the program. This is ok though, as either way acheves the same result for what I’d like any general person viewing the code to see, the only issues are the readability in the future, and general speed of the program, which is barely a problem for a code with such a small set of requirements.