**Controlled Assessment**

**Planning**

The code will allow the user to play a game of top trumps against the computer. Cards will be based on celebrity dogs and their attributes, which will be imported from a file called cards.txt. The computer will handle the comparison of cards and movement of the cards. The user and computer will take turns to choose a category to compare- the winner of the round will be the person with the highest stat. The winner will take the opponents card and put it to the bottom of their pile. The game will repeat until one of the players loses by having no cards left.

Each card will be imported from a predetermined list of names that are stored in a file called dogs.txt. The cards will have 4 attributes each. There can be between 4 and 30 cards in the game and the cards are randomly arranged in the code.

The dogs attributes will be randomly generated using the random.randint function with the ranges given below.

Drool (1-10) Lower is better

Exercise (1-10) Higher is better

Friendliness (1-100) Higher is better

Intelligence (1-100) Higher is better

The category that will be compared will always be selected by the player for the first round, but then it will be selected by the winner of the last round for every concurrent round.

String and array manipulation will be used to move cards around. The card that is to be used in the game will always be the first card in the array (index 0).

A 2 dimensional array will be used to store the player’s card. This is probably the most efficient data structure to be used for the purpose as it allows very easy access and manipulation of the data in the array. The array is also suitable as all data to be stored in the array will be of similar structure.

Functions and global variables will be used to appropriately split the code into parts based on its function, to allow the code to easily be repeated as needed, simplifying the code.

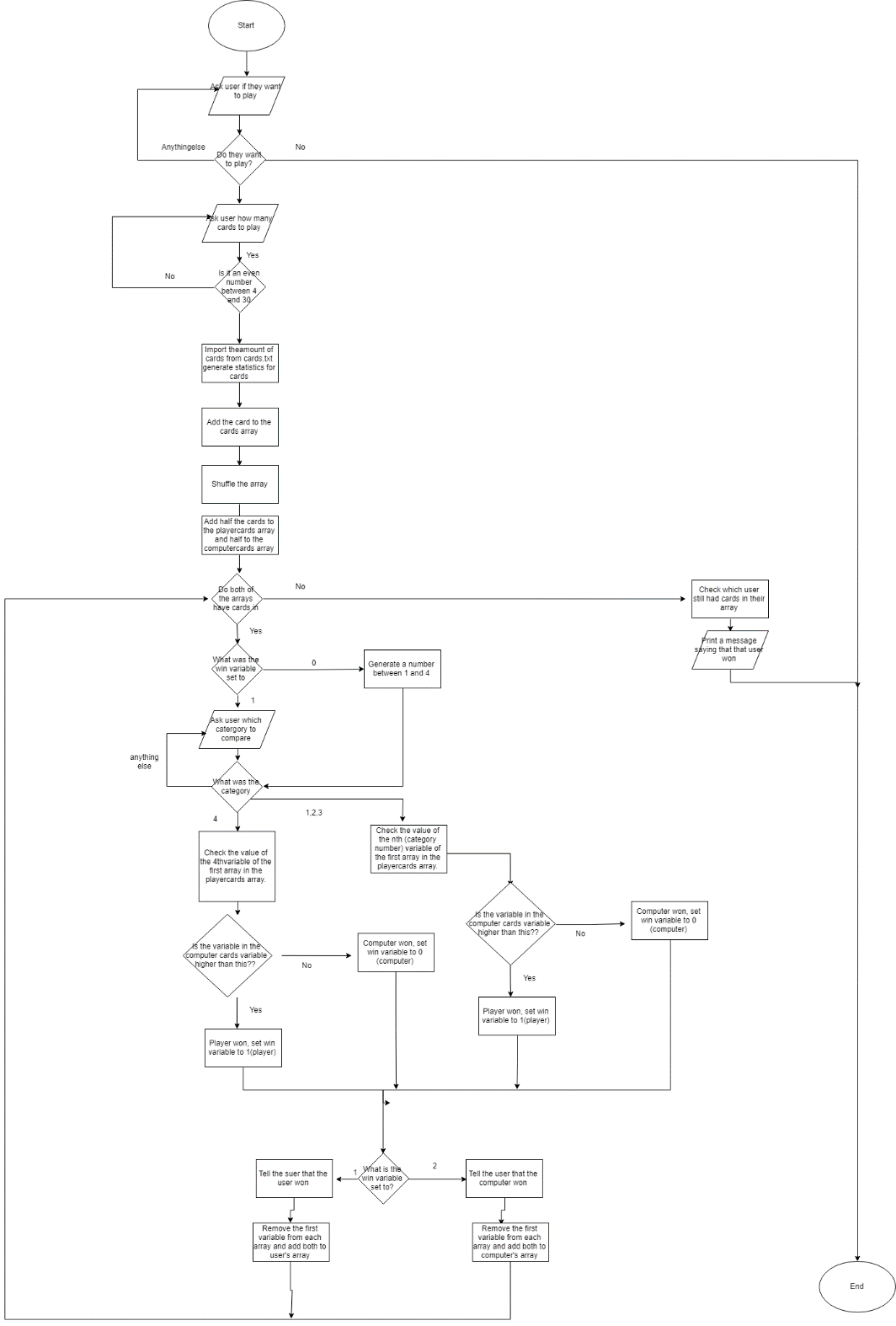
Iteration will also be used throughout the code to appropriately repeat the code as required and also to reduce the amount of selection needed.

Throughout the code, input validation will be used to reduce the occurrence of errors and I will use selection and string manipulation to allow a wide range of inputs, so the code should work fine

Some exceptions were needed in the code which required additional code, such as the treatment of the drool category, which is treated differently to the other.

To test, I will change the cls() function to only print 2 lines and will use the print function to display the contents of variables and lists, as required.

I decided to only use a flowchart to plan my code, as opposed to using a combination of the flowchart and some pseudocode, as the task was relatively simple, as well as the fact that python is very easy to read anyway.

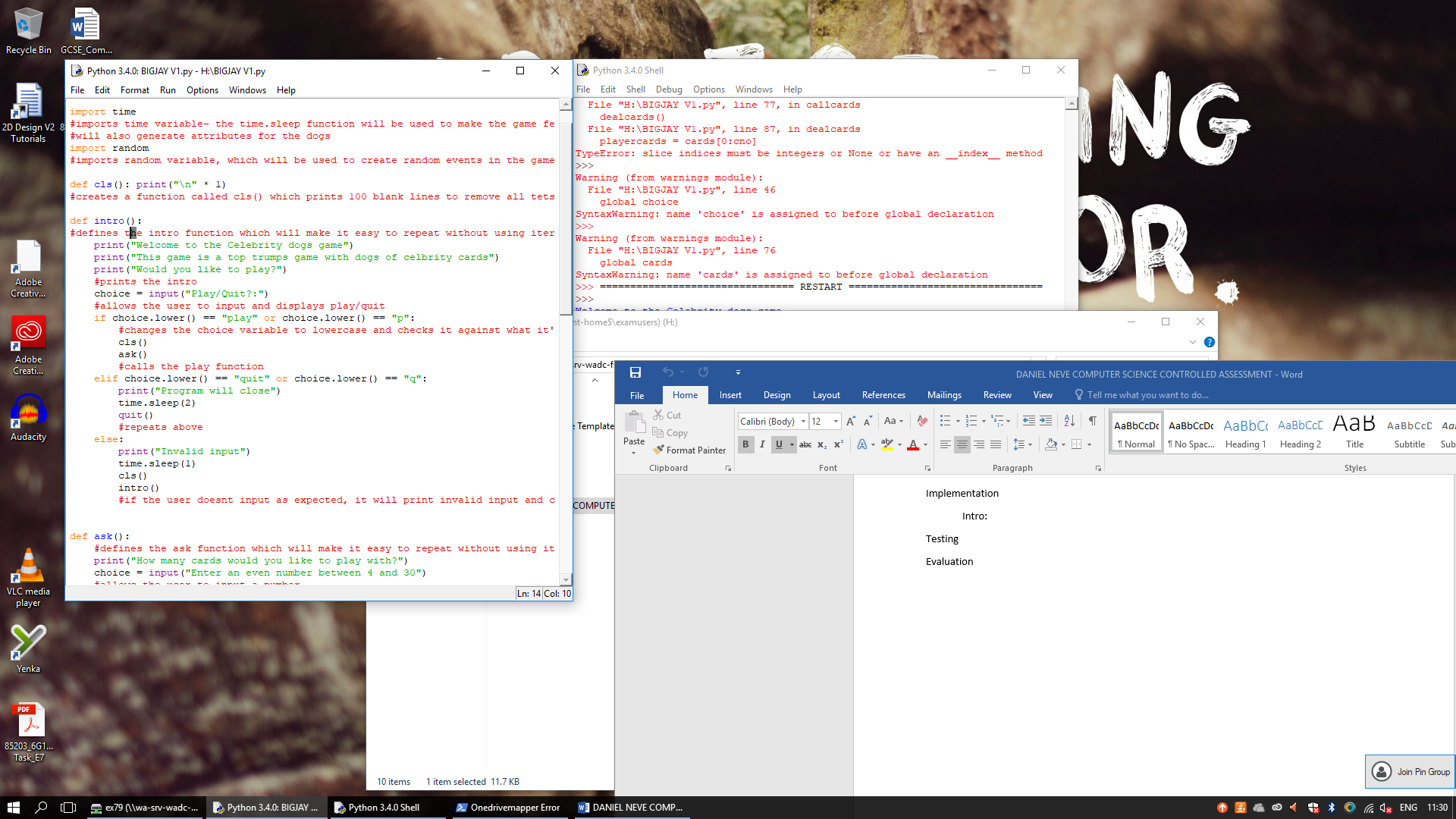
**Flowchart**

**Testing:**

I will test each function individually, to ensure it is receiving the correct data from other functions and to ensure that it works as expected. I will play the game many times using 2 different types of data – normal data (data that will be expected during normal gameplay) and error data (data that could be out of range or has invalid characters- to ensure that the program correctly handles the gameplay.

**Implementation**

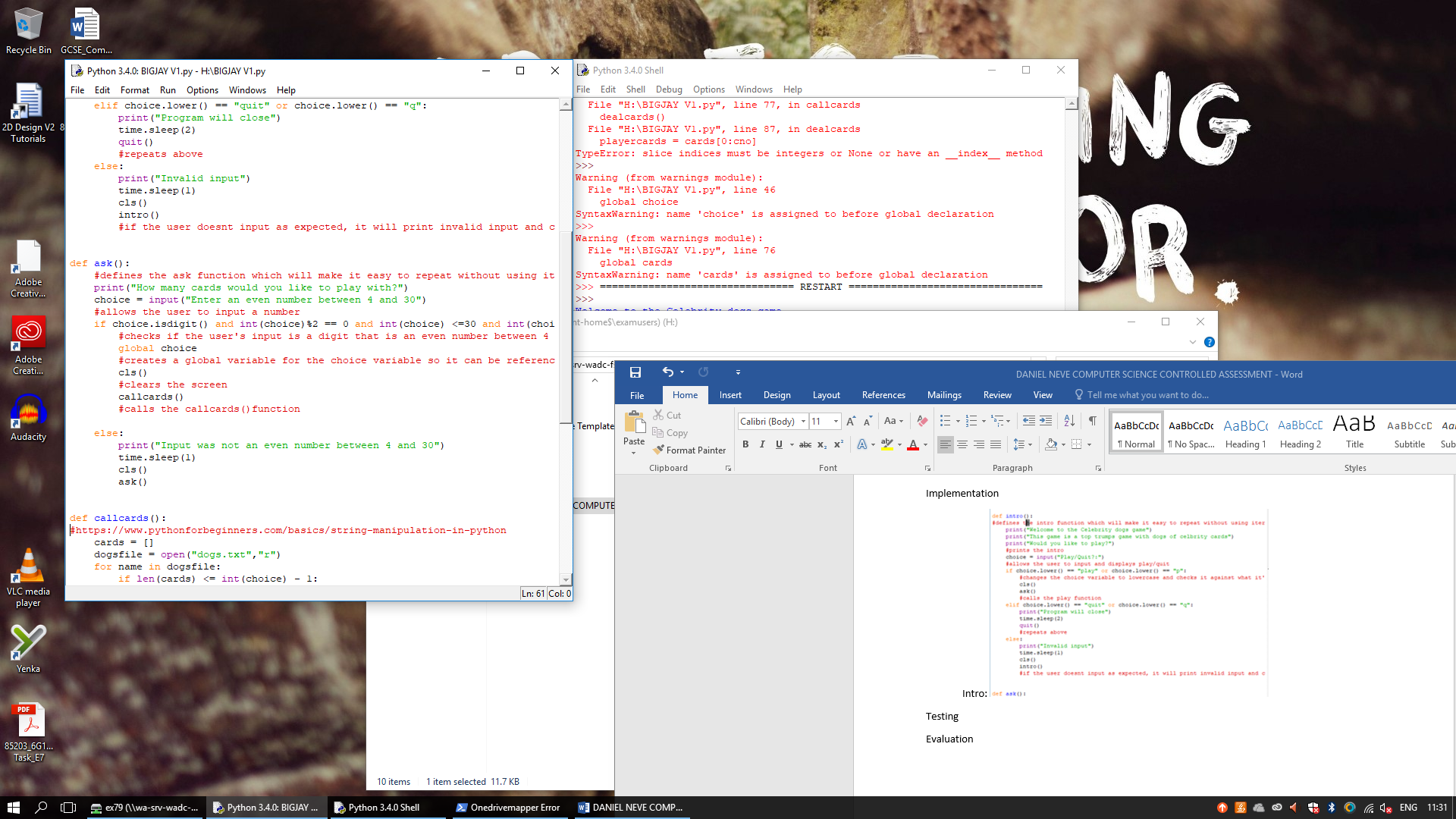
Intro: (Task 1,2,3)



The intro function completes task 1,2 and 3. It introduces the user to the game and then gives them the option to play and quit. If they put quit, the program exits using the quit function. If they enter playgame it will call the ask() function to ask them how many cards they want. It has error checking and will loop until a valid input is received. It also has code to allow a variety of inputs to reduce the chance of an error. The time.sleep() function is used to slow the code down and the cls() function is used to remove all text from the screen.

Testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function: | Testing for | Expected Input | Input | Expected Output | Output |
| Intro | Ensuring that the string manipulation allows a wide range of valid inputs and to ensure that errors are correctly handled | Play,p,quit,q any case | play | The code will continue, clear the screen and call the ask() function |  |
|  |  |  | P | The code will continue, clear the screen and call the ask() function |  |
|  |  |  | Quit | The game should print a message, wait a few seconds and then quit |  |
|  |  |  | q | The game should print a message, wait a few seconds and then quit |  |
|  |  |  | PLAY | The game should print a message, wait a few seconds and then quit |  |
|  |  |  | yes | The game should print invalid input, wait a few seconds, clear the screen, then return to the menu |  |
|  |  |  | 1 | The game should print invalid input, wait a few seconds, clear the screen, then return to the menu |  |

Ask: (Task 3)

The ask function is for task 3. It will ask the user how many cards they would like to play with. It then checks the user’s input to make sure it is an even number between 4 and 30. It will loop until a valid input is received. It will then call the callcards() function and create a global variable called choice, so it can be referenced later.

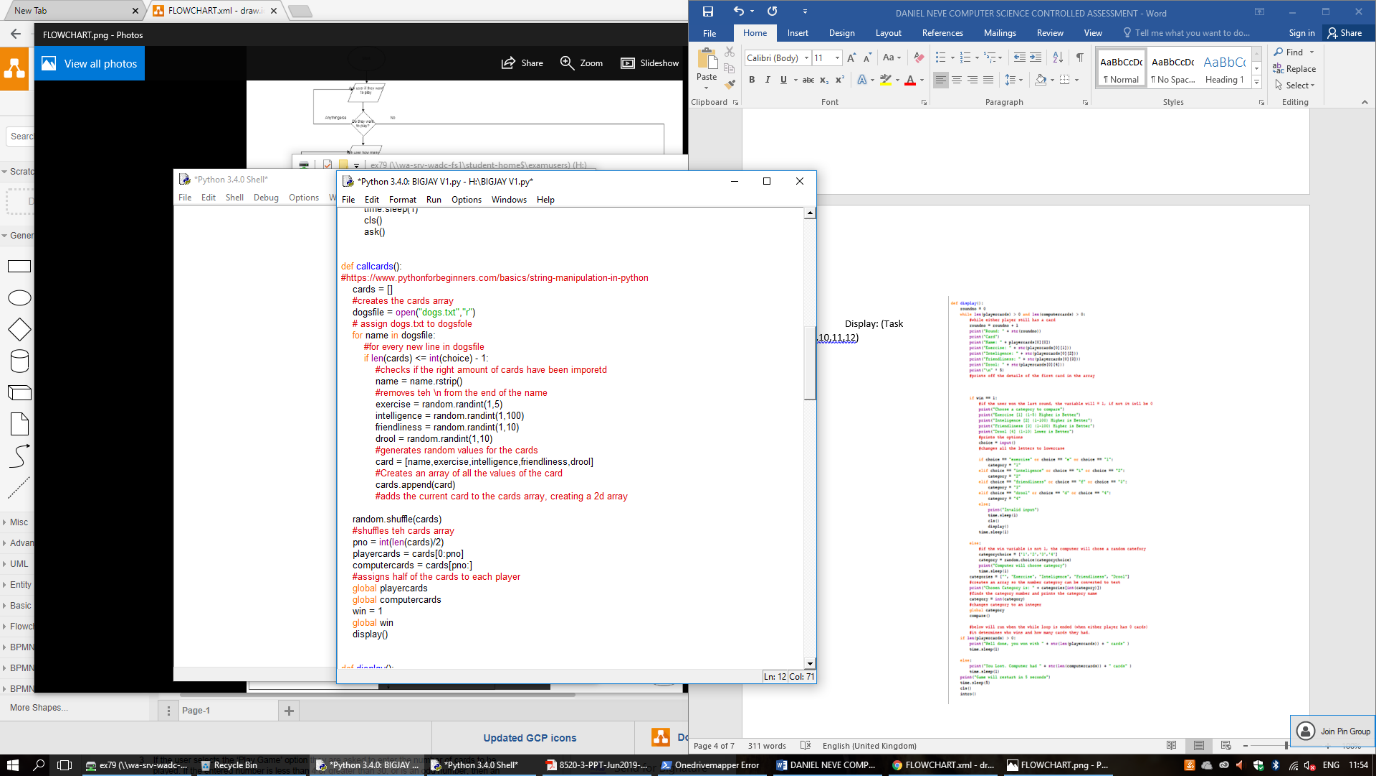
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function: | Testing for | Expected Input | Input | Expected Output | Output |
| Ask | Ensuring that the string manipulation allows a wide range of valid inputs and to ensure that errors are correctly handled | An integer between 4 and 30 | 4 | The code will continue, clear the screen and call the callcards() function, crfeting a global variable for the coice variable |  |
|  |  |  | 30 | The code will continue, clear the screen and call the callcards() function, crfeting a global variable for the coice variable |  |
|  |  |  | 2 | The game should print an error message, and then ask the user again for a number. |  |
|  |  |  | 31 | The game should print an error message, and then ask the user again for a number. |  |
|  |  |  | x | The game should print an error message, and then ask the user again for a number. |  |

Call Cards: (Task 4,5,6)

I am going to use a 2 dimensional array to store the cards. It is a simple data structure that allows easy manipulation of the cards, which will make the code for later on much simpler. It will also allow a very easy shuffle feature, while ensuring all the cards keep the statistics they were assigned initially.

The 2D array will look like this this:

Cards [x][0] will always give the name of card [x], while card [0][x] will show the x value of the 1st card (index 0)



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cards array index no(Primary Array) | | | | | | | | | | |
| **Dog array index no (Secondary array)** |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **(Upto 29, based on how many cards user selects)** |
| **0** | **name** | **name** | **name** | **name** | **name** | **name** | **name** | **name** | **name** |
| **1** | **Exercise** | **Exercise** | **Exercise** | **Exercise** | **Exercise** | **Exercise** | **Exercise** | **Exercise** | **Exercise** |
| **2** | **intelligence** | **intelligence** | **intelligence** | **intelligence** | **intelligence** | **intelligence** | **intelligence** | **intelligence** | **intelligence** |
| **3** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** | **Friendliness** |
| **4** | **drool** | **drool** | **drool** | **drool** | **drool** | **drool** | **drool** | **drool** | **drool** |

This part of the code is used to import the dogs.txt file as read only and assign it to the dogsfile variable. It then uses a for loop (for every line in dogsfile), as well as an if loop (to check how many cards there are) to create the number of cards the user selected before using names from dogsfile. The statistics are generated using the random.randint function and then they are the cards (which are arrays) are stored in a larger array called cards, which forms a 2 dimensional array. The program then uses the random.shuffle function to rearrange the cards array. It then divides the number of cards in the list by 2 and assigns half to the playercards array and half to the computercards array. It then creates global arrays so they can be referenced later. It also sets the global win variable to 1 so the user will select the first category. It calls the display function.

To test this function, I will use the print function to show various different parts that the program is creating, as it will not have any user interactions normally. I also validated that the program added the correct amount of cards- the cards the user selected. The random.shuffle function worked well too, and an equakl amount of cards was distributed to each user. I also ran many times to make sure that the generated statistics were in the ranges specified. For testing, I will aslo remove the refence to the display() function.



I used 4 cards to check that it worked as expected. It adds a new card to the cards array every time it repeats until the length of the array is equal to the number of cards the user selected.

It is repeated 30 times, as it repeats for every line in the file.

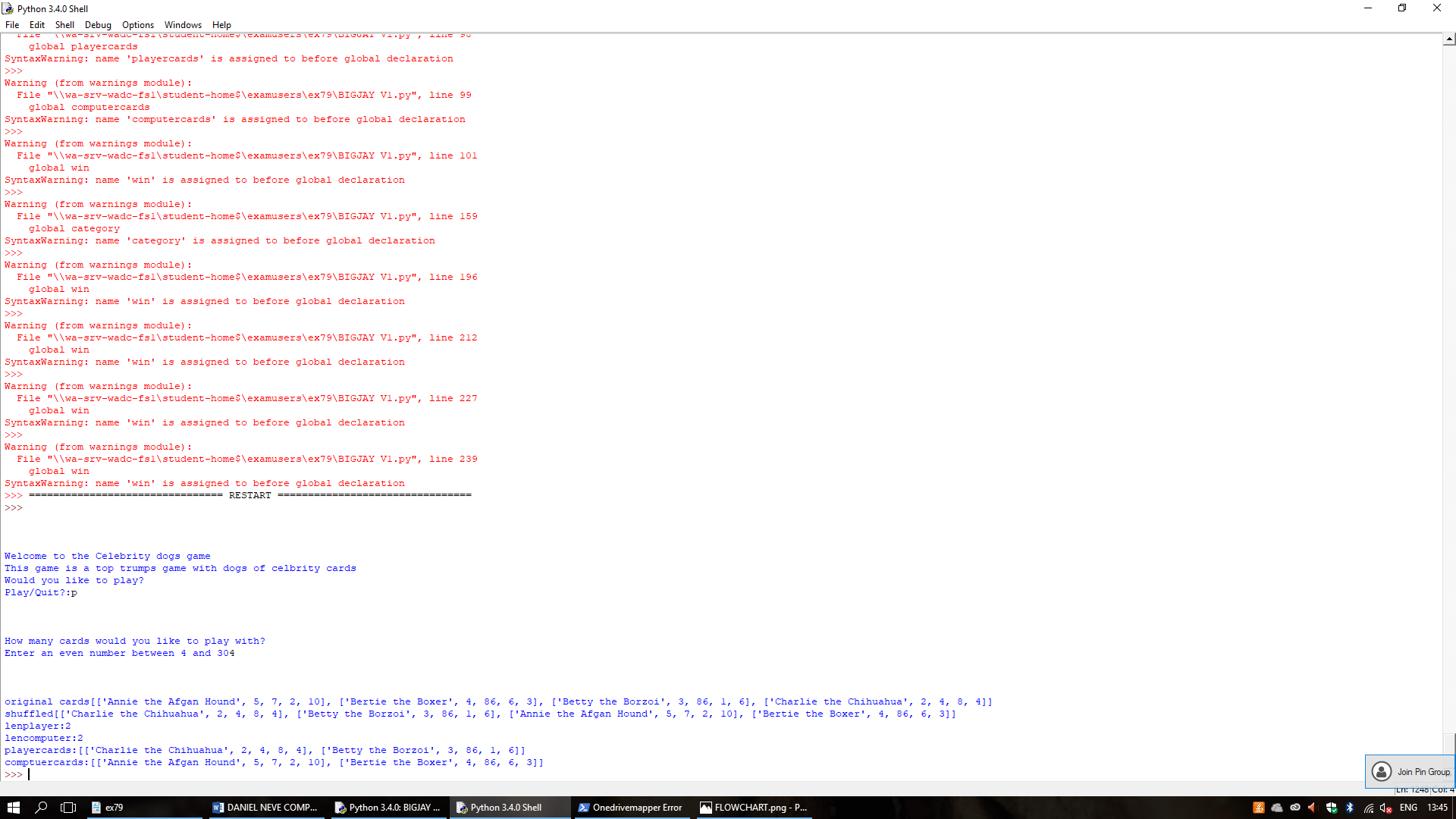
I used string manipulation (.rstrip) function to remove the new line markers from the name variable.

The numbers generated were all in the ranges needed.

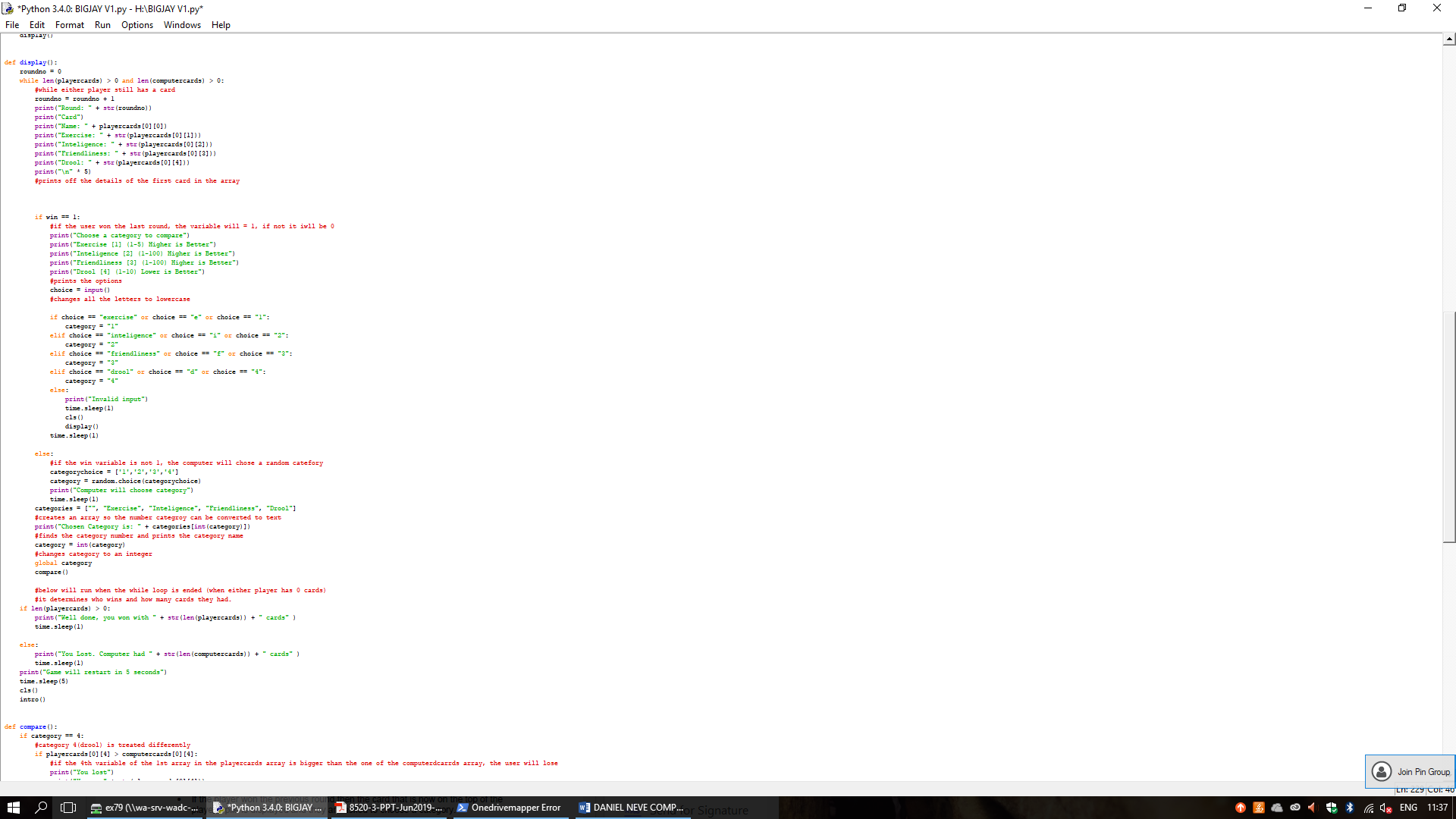
I have also completed the same test using 30 as the number of cards. All the cards were different and all ranges were as expected- I changed the code so it printed one card per line.



Now that I had verified all of the cards were added correctly, I needed to check that the shuffle function and the function to split the cards into two sets.



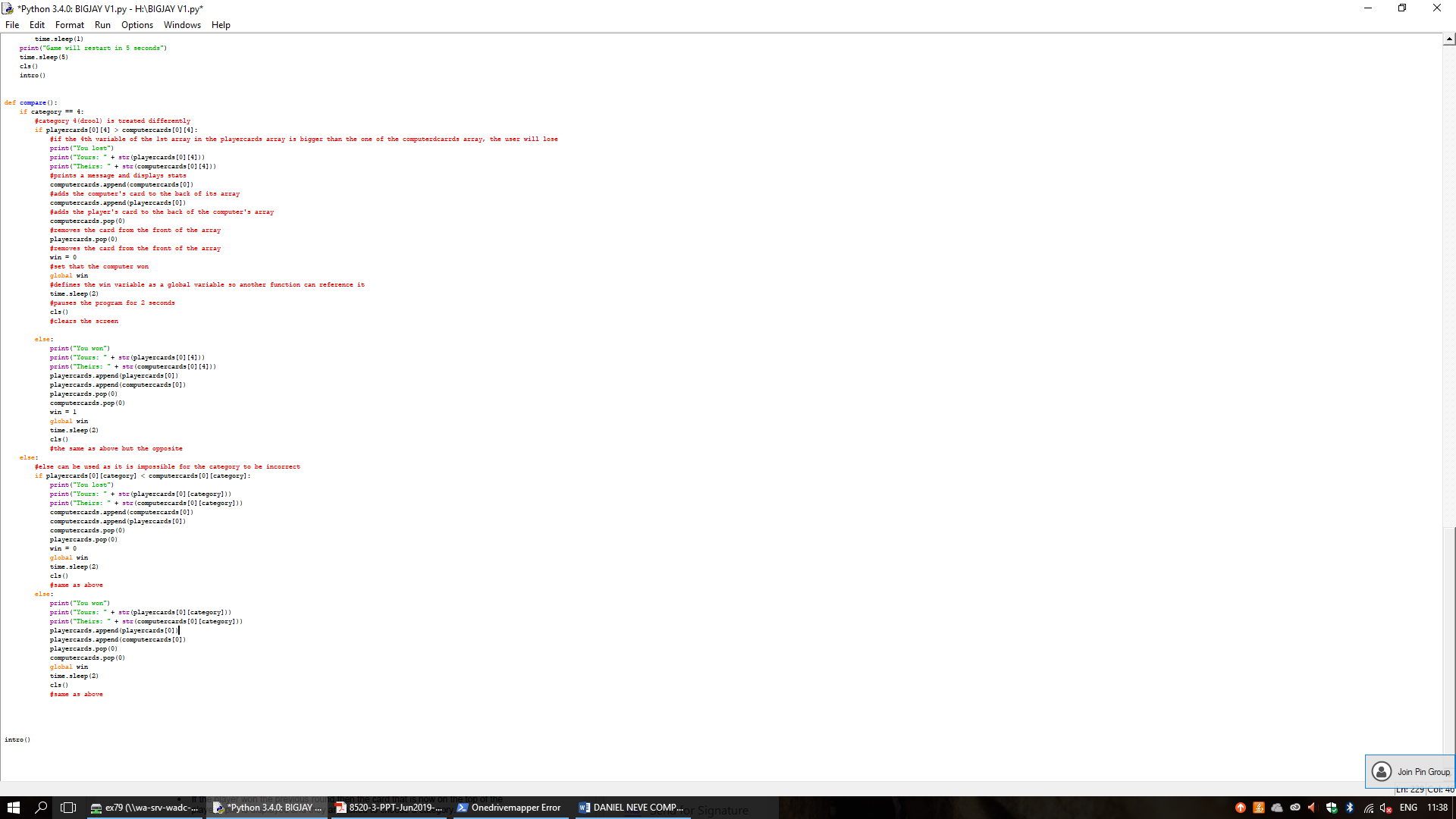
The cards in the shuffled pile were in a different order to the cards in the original pile and the cards were equally split between the cards- 2 cards were assigned to each pile. As well as this, the cards retained the correct values that they were assigned, so the 2d array is working correctly.

Display: (Task 7, 10, 11, 12)

The display function will receive the player cards and computercards arrays as well as the win variable from the previous function. It then sets the round number to 0. It creates a while loop that will run until one of the arrays have no cards in them. Every time the code repeats, it adds 1 to the roundno variable.

It then prints the details of the cards. The card displayed will always be the 1st position in the array (index 0)

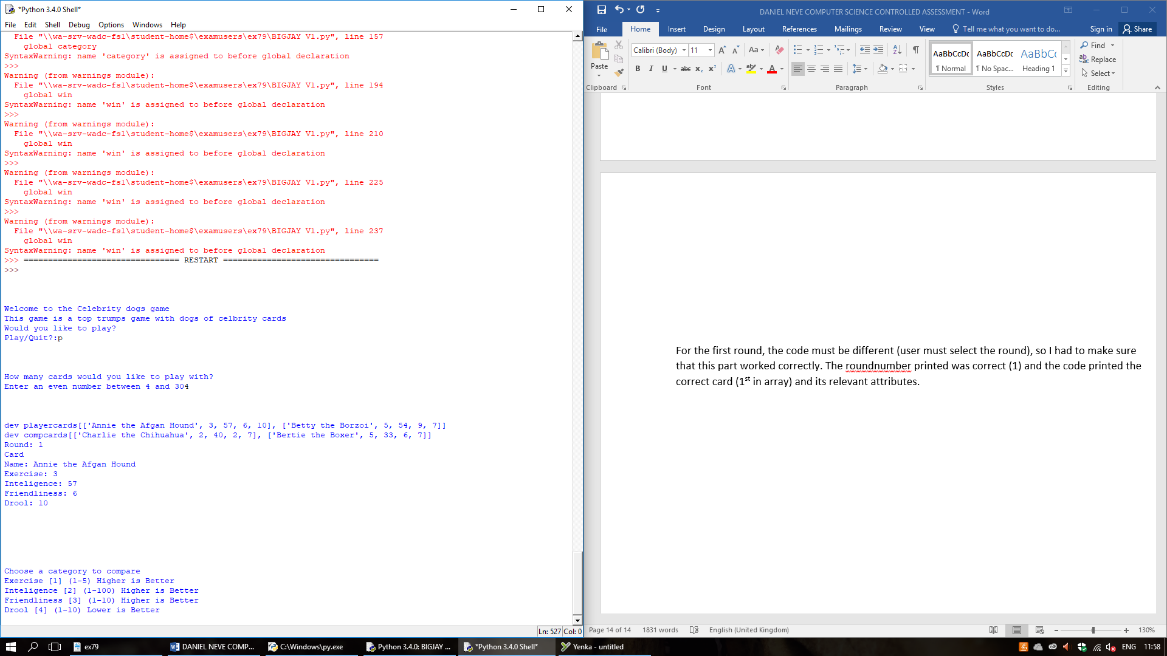
I realised while testing that I had written the wrong thing to print (friendliness is between 1 and 10, not 1 and 100). I also decided to just test display and compare together as they are both reliant on each other and were only split to allow simpler iteration and to simplify the code.

Compare: (Task 8, 9, 10)

The compare function will receive the category choice from the display() function. It checks the category choice, as category 4 must be treated differently. Its function is explained in the annotations on the code. It uses the .pop () and .append () functions to remove the cards from the front of the piles and add them to the back of the winning player’s array. It also sets the win variable to 0 or 1, based on who won the round, so the right player chooses the category.

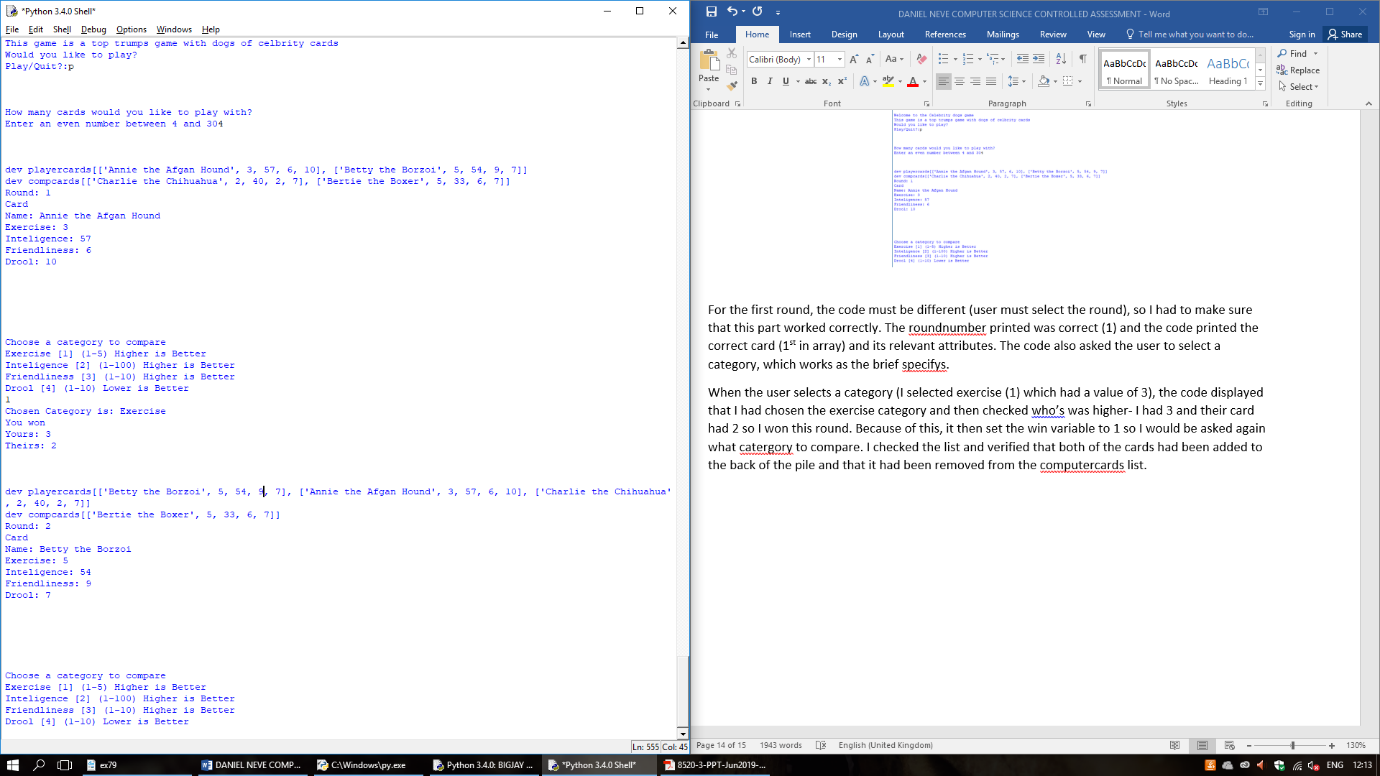
I will test the display and compare functions together as their function is dependent on the output of the other- they were only split to allow for easy iteration and to simplify the code structure.

To test this, I played the game as normal to test the two scenarios which would change the outcome: user wins and computer wins. I also used additional print statements to make sure the cards were moving correctly. I also needed to check that the game ended at the correct time and restarted as it should have. I will use 4 cards for all the testing process.

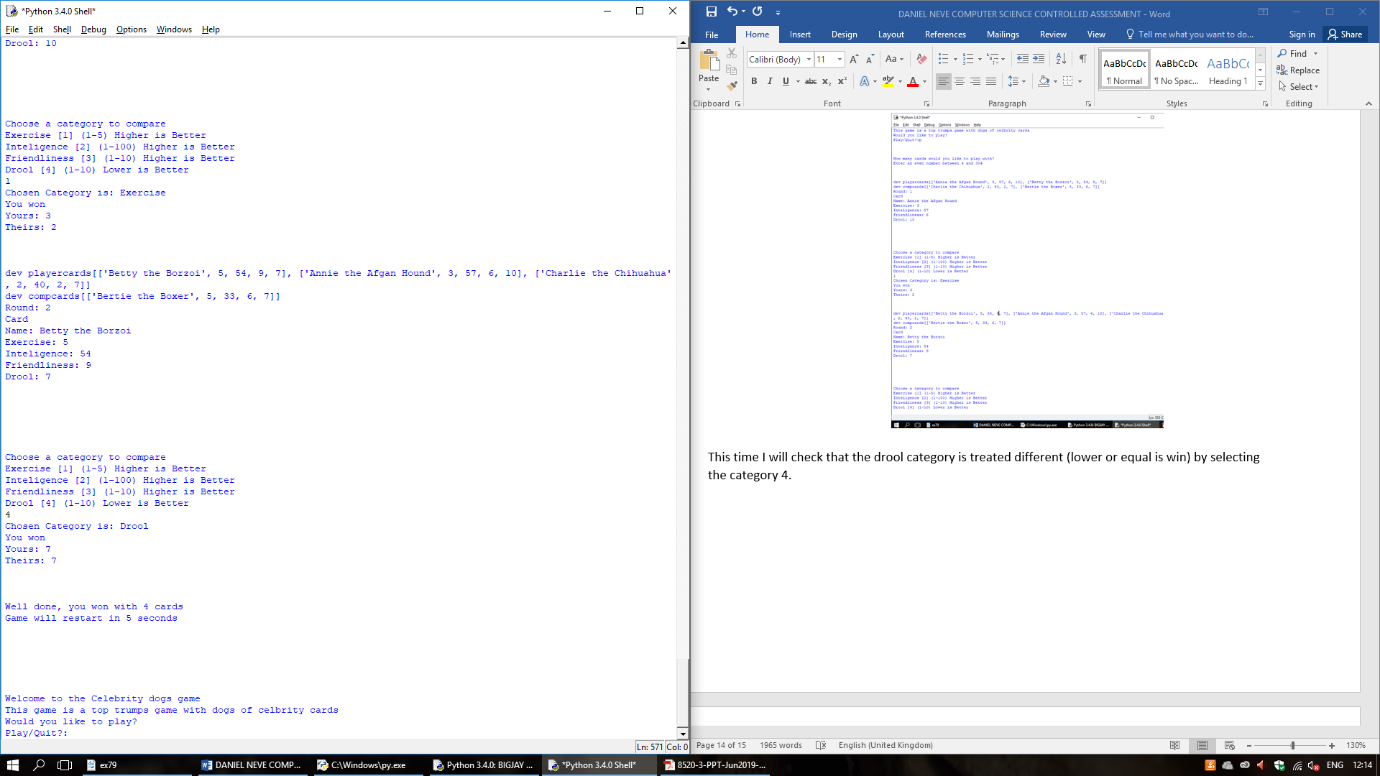


For the first round, the code must be different (user must select the round), so I had to make sure that this part worked correctly. The round number printed was correct (1) and the code printed the correct card (1st in array) and its relevant attributes. The code also asked the user to select a category, which works as the brief specifies.

When the user selects a category (I selected exercise (1) which had a value of 3), the code displayed that I had chosen the exercise category and then checked whose was higher- I had 3 and their card had 2 so I won this round. Because of this, it then set the win variable to 1 so I would be asked again what category to compare. I checked the list and verified that both of the cards had been added to the back of the pile and that it had been removed from the computer cards list.

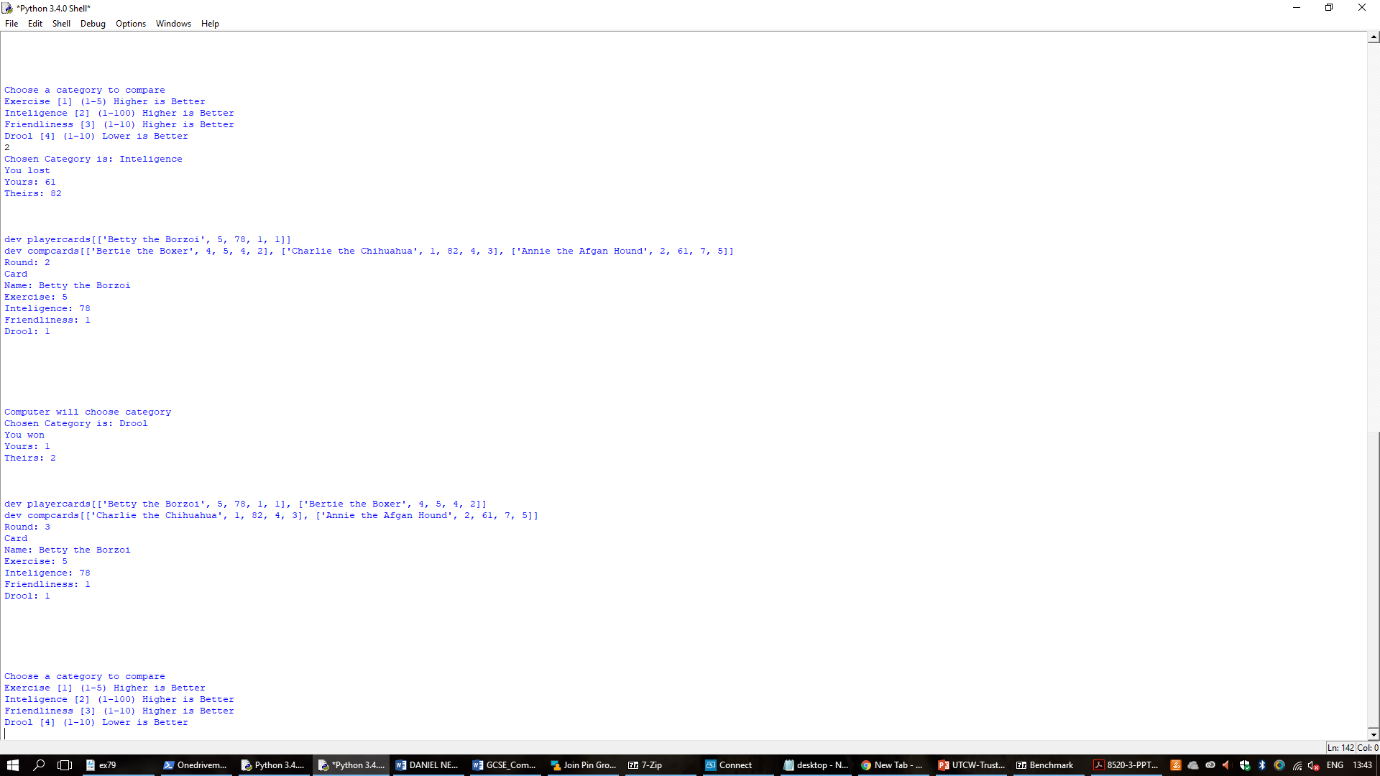


This time I will check that the drool category is treated different (lower or equal is win) by selecting the category 4.



The code made me win the game because my drool statistic was equal to the computers. Because I had won both rounds, I had all of the cards, so the game detected that I had won.

I started another game and tried to check what would happen if I lost both of the rounds, I used the same process as before.



The game detected that I had lost the round and removed the card from my list and added both cards to the back of the computer cards array. It then continued to display my card as it should have, before selecting a random category to compare the next round. It chose drool- I had 2 and the computer had 2, so I won. The game then added the cards back to the array and repeated.

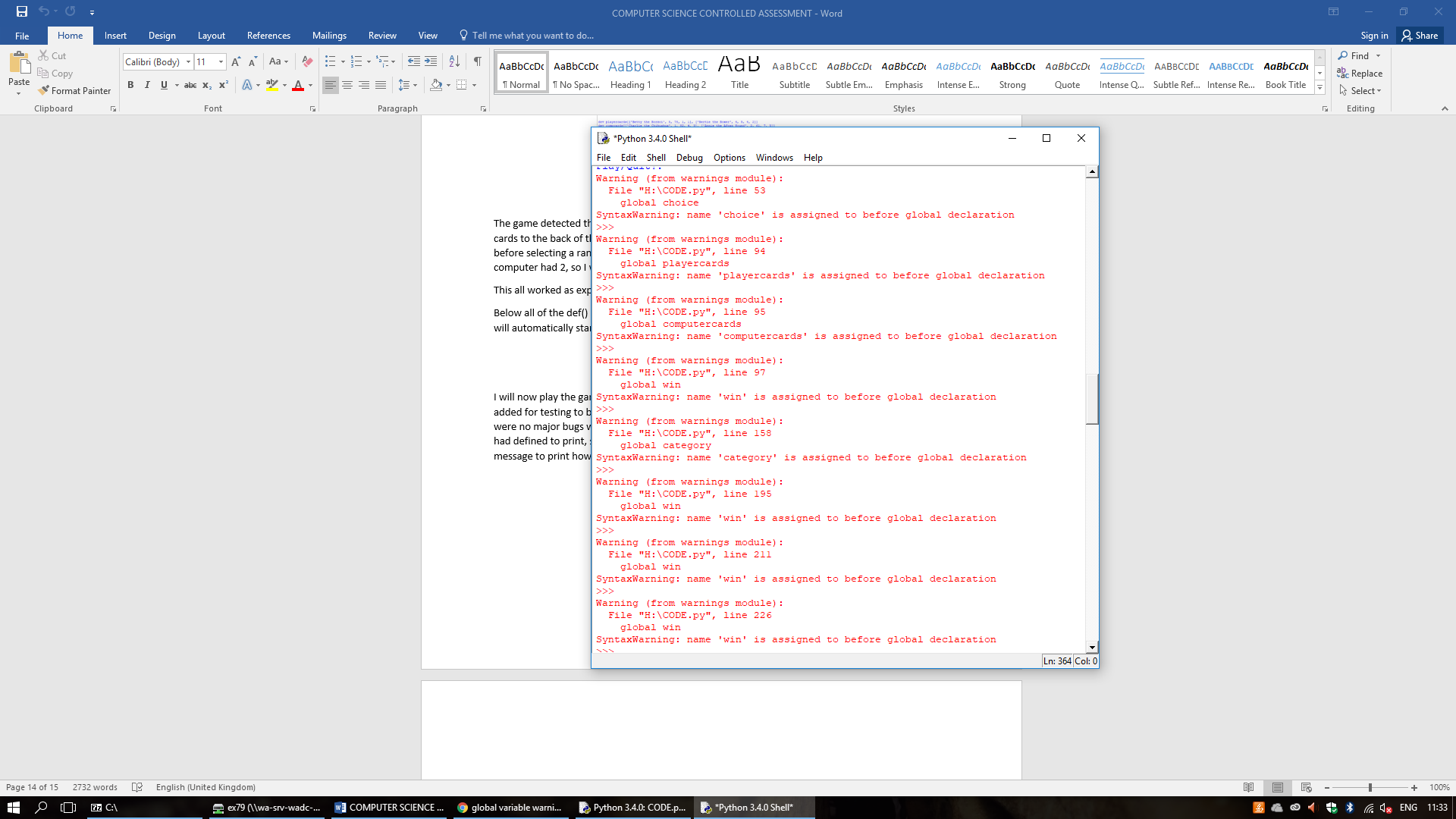
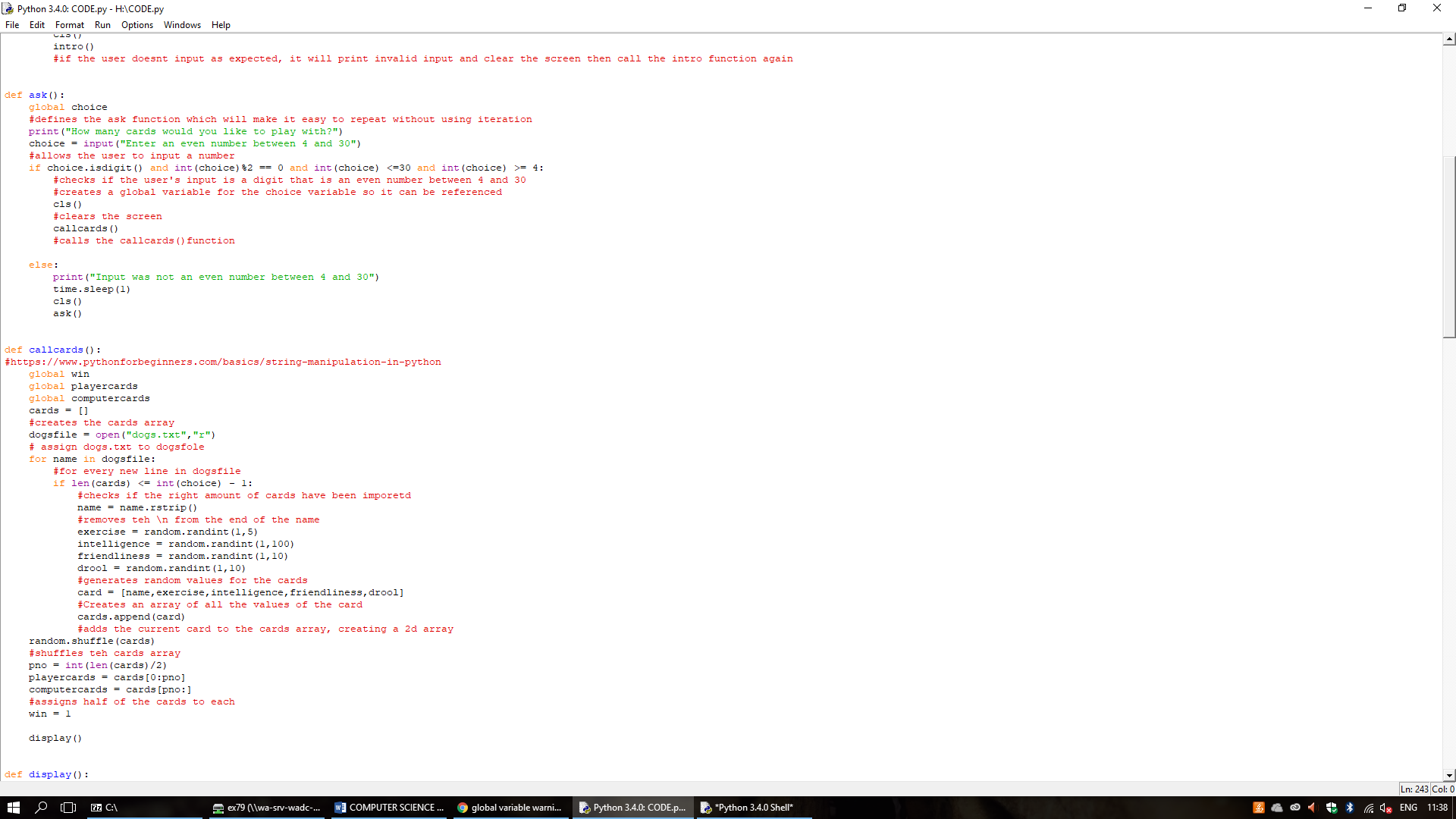
This all worked as expected.

Below all of the def() statements, there is a command to call the first defined function (intro). This will automatically start the game and work through the code as expected.

**Testing of the whole game:**

I will now play the game with a larger number of cards (8) and remove the print statements that I added for testing to be able to see if there is any bugs that were not visible straight away. There were no major bugs with the code, but I realised that there were some issues with the messages I had defined to print, such as the category numbers being wrong. I changed these and also added a message to print how many rounds the user had played when the game ended.

I later found that I had issues with global variables, where the error “name (variable) was referenced before global declaration”. I found that this was caused by me placing my global declarations towards the end of the function code blocks, rather than at the start. I rectified this by moving all of my global declarations to just below the def() line for each block. This fixed the error, so I tested to ensure that the code was fully working correctly still, using the testing methods used before and it worked fine. I also found that, because I was using functions, I could not just put the global declarations at the top of the code with the import statements. Really, this did not affect the functionality of the code and was only rectified to prevent there being errors in the code.



**Evaluation:**

For the task specified, the code worked very well. There were no bugs and the code matched all of the specified criteria. There are many different changes that I could have made to refine the code or to make it work better if the scenario was different:

If the cards were different and had attributes that could be different between the cards, such as special abilities, I would probably use a custom data structure, such as a class, to store the data. Because the attributes on the card are the same for every card, a 2D array was fine for this purpose.

I could also have condensed the code by using iteration and selection to print the messages and move the cards between piles, as opposed to writing the code 4 times and changing the parameters. This would have added some more complexity to the code, but would have allowed me to remove the unnecessary repeated code. Despite this, I think that the code has a good level of efficiency and has good use of both definite (conditional) and indefinite (unconditional) iteration.s

The code does very easily handle invalid inputs and also allows the user to input a wide range of inputs and still handle the string as it should be. This makes the code very robust against most human error: It allows any case to be inputted and can handle errors easily using iteration, without causing issues.

To improve the quality of the game when replayed, I could have randomised the order of the cards before they were assigned attributes and then added to the cards array. This is because the cards are currently imported in the order that they appear in the dogs.txt file so, when a set number of cards are selected, the cards in play will always be the same. This isn’t a massive issue as the statistics of each card are different every time and the cards array is shuffled to ensure that the user doesn’t always end up with the same cards.

The UI could also have been improved to improve the quality of the game- I could have used either better text formatting to make the game look better as it was, or I could have implemented a GUI using the GUIProgramming module. This was not necessary for the task and the UI design I implemented, and using the CLS() function provided a playable game.