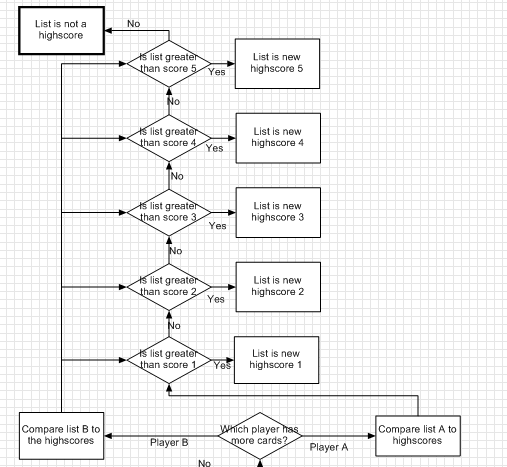
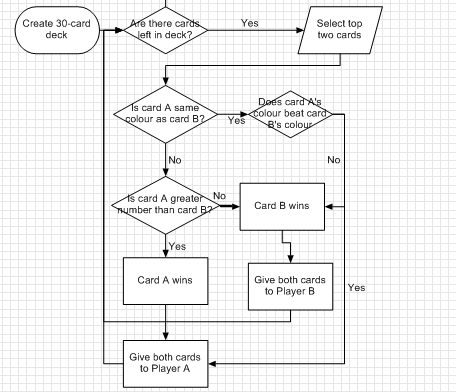
GCSE NEA Computer Science Exam

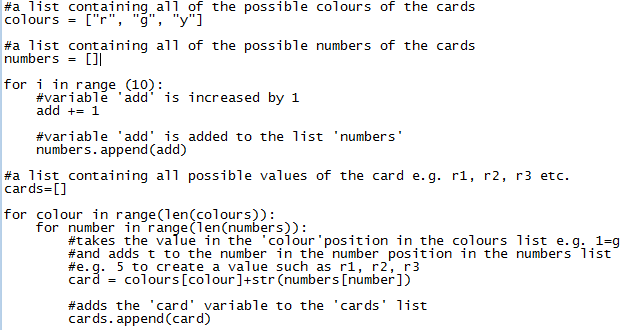
Algorithm



Requirements Breakdown

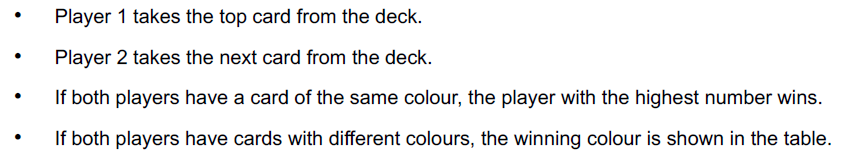


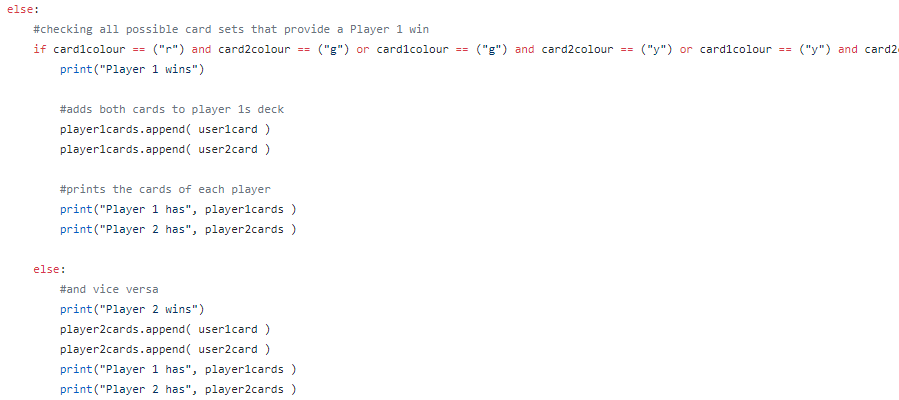
The code needs to have 30 cards with 10 numbers for each colour. This is best achieved with two lists and a FOR loop.

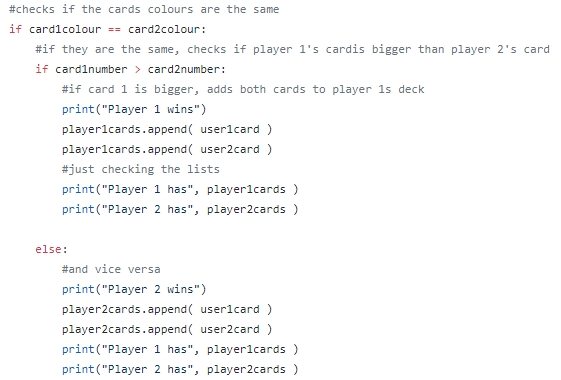




The shuffle can be done using the random module two ways, the most efficient is random.shuffle(list). This will code will simply shuffle the values in the list into random positions.

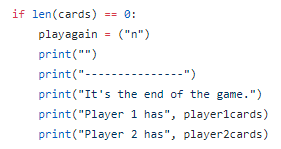


After shuffling, all I have to do is take the two top cards and then remove them from the list. I compare the colour first and then the number.

First of all I check if the card colours are the same, if they are then I compare the card numbers. The card with the biggest number wins and the winner gets both cards.

If the cards have different colours then I first check to see if card 1 wins, if card 1 doesn’t win then card 2 does. The winner gets both cards.

Red > green > yellow.



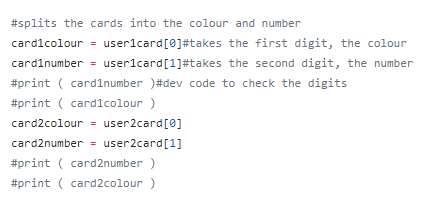
If there are no more cards in the deck, I set playagain to n so that the code no longer runs and give a display of all of the cards that each player has.



After the game is over, I check which player has more cards and then add the winner’s name to the list so that it can be added to the scoreboard later. I also display the difference between the two player’s decks.



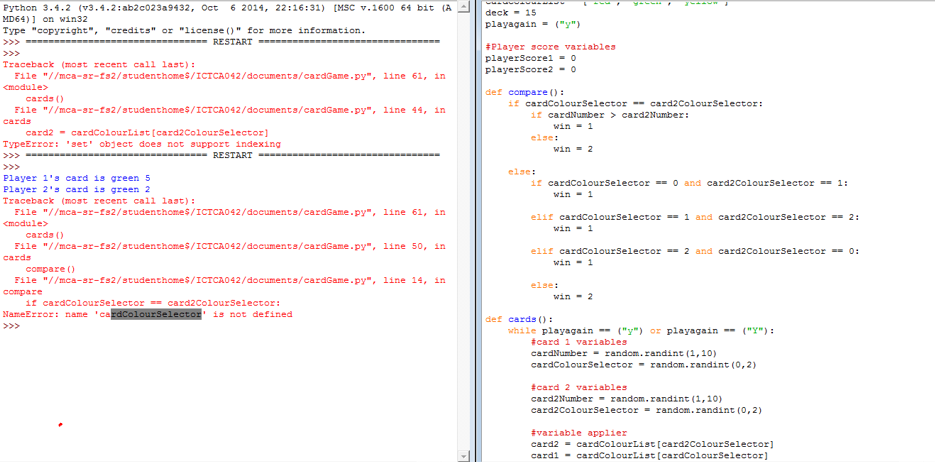
If there are still cards in the deck then it asks if the players want to go again.



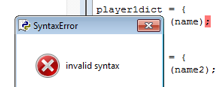
When I was comparing the cards I had to split them into the colour and number, I separated the number by creating a variable that is equal to the second digits of each players’ cards .I then did the same for the first digit to get the colour.

I have not managed to get a script working that allows a scoreboard to be implemented.

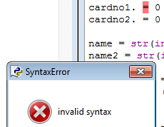
Errors



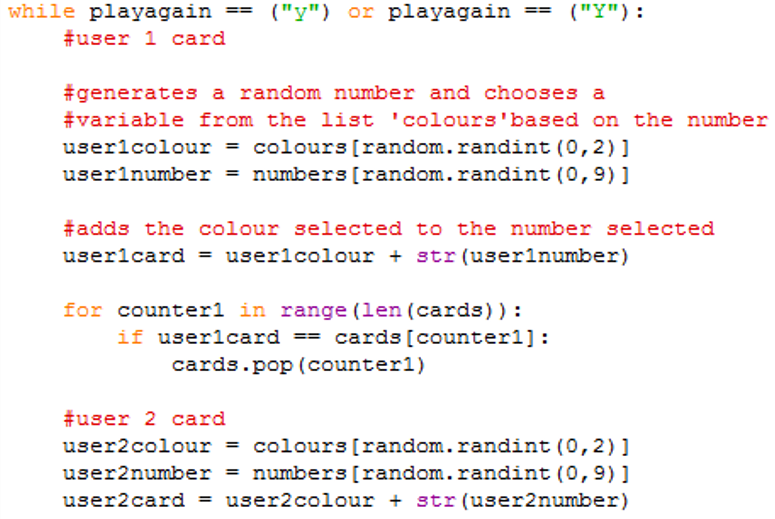
My first error occurred in the first version of my game where I tried to use defined scripts to perform all of the tasks. This didn’t work because the variable was called before it was defined. This error is why I switched to a non-definition version.



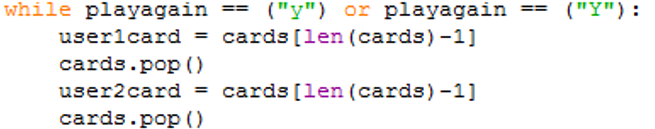
The second error was a syntax error, I accidentally used semi-colons instead of colons.



Python doesn’t like full stops.

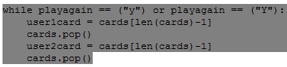


I shuffle the cards and then I select a random card, this is pointless because the deck has already been shuffled so there is a random card on the top. Selecting a random card from a deck of randomly shuffled cards is just pointless and makes the code needlessly complicated.

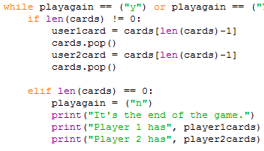


By just shuffling the deck and selecting the top two cards the code is simpler, faster, and easier to read.

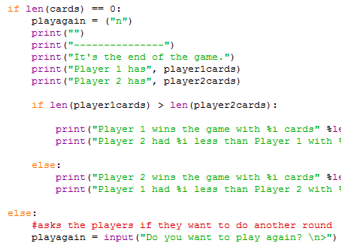




An index out of range error occurs because there is no way to stop the code when the deck is empty, because I don’t check to see if the deck is empty the game tries to keep taking cards when there are no cards to take. I can fix this by simply checking the length of the list, if the length is 0 the deck is empty and I can stop the code.



I added a piece of code that checks if there are still cards in the deck, if there are no cards then it stops running the main code block and tells the players what cards they have. However, I put this code right after it selects the cards and after the game has started running again so the players get ready to play another round only to be told that they can’t, pretty much pulling the rug out from under them.

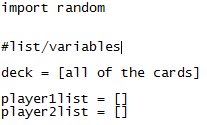


By moving the deck’s length check to the end of the code I can integrate it better into the replay system. Now the game only asks the players if they want to play another round if there are cards left in the deck. If there aren’t then it sets the variable playagain to n so that the main block doesn’t start again. The players are never asked if they want to play again before being told they can’t.

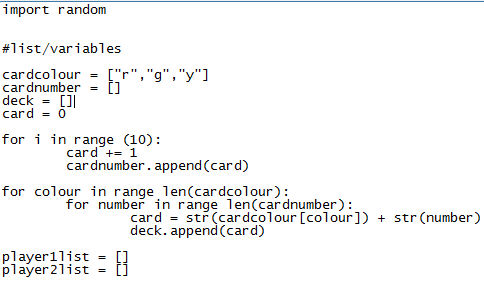
Development

The first thing I did was figure out which modules I’d need and how many variables I needed. I luckily only needed the random module for the code, I knew I would need lists to hold the player’s cards and to store the deck.

I imported the random script and then created the lists I’d need, I gave the lists names that should make it easy to understand their purpose just by reading it.



The next thing I did was create the lists to hold the values for the cards, that’s the colours and numbers. I chose the colours red, yellow, and green and the numbers 1-10. Because I’m lazy I didn’t want to manually type in every number so I created a loop that would do it for me.



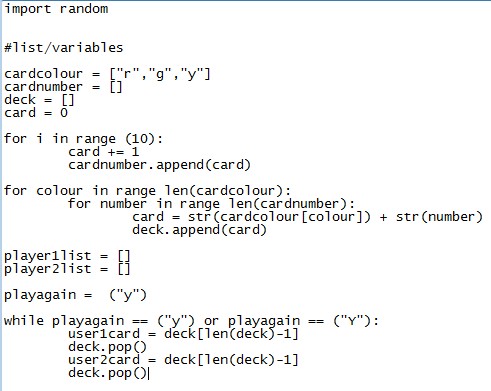
After I had all of the base values, I had to combine them into the cards so instead of a load of unrelated numbers and unrelated letters I’d have numbers and letters joined together to mean something. I did this by looping through all of the colours and looping through all of the numbers and adding them together to get a single value E.G. R1, Y5, G9. After the list creates a card, it adds them to a separate list called deck.

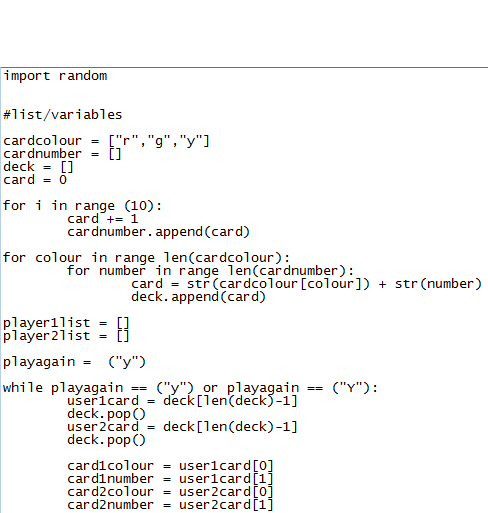
Next, I had to shuffle the deck so that the cards would all be random. I originally used a random number generator to generate a number from 0-29 and selected the card in that index of the deck. However this was complicated and difficult to read, very un-Pythonic, so I switched to just using the random module to shuffle the deck.

After this I moved on to the main block of code that would actually select the player’s cards, compare them, and decide the winner. But first I make sure that there is some user control by using the variable Playagain, later on I program a check that asks the player whether they want to play again or not. For now I just have the check though, therefore if the variable Play again == “Y” only then will the code run.

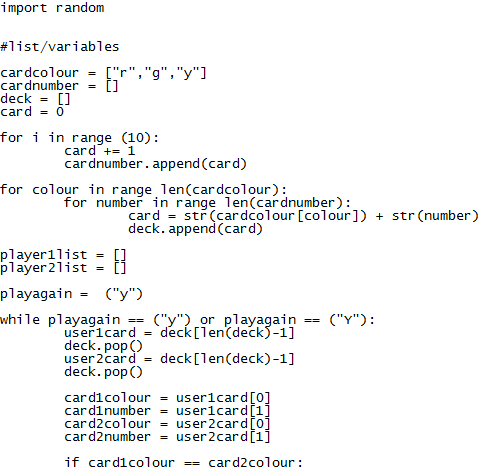
Next I need to make sure that the players receive a card, because I’ve already shuffled the deck the top two cards are random so I can just select them. To select the top two cards I chose to do User card = deck [len(deck)-1]. If there are 30 cards in the deck and I chose the 30th I would actually have to do deck[29] because Python counts from 0 so the first value is considered to be deck[0] and every other value is deck[x-1].

After the players have been given their cards, I need to remove those cards from the deck otherwise they’ll stay the top cards and the players will get the exact same card every time. To prevent this I can use the Pop command, if there is no index specified it will just delete the uppermost value – the cars that have just been given away so I do deck.pop().

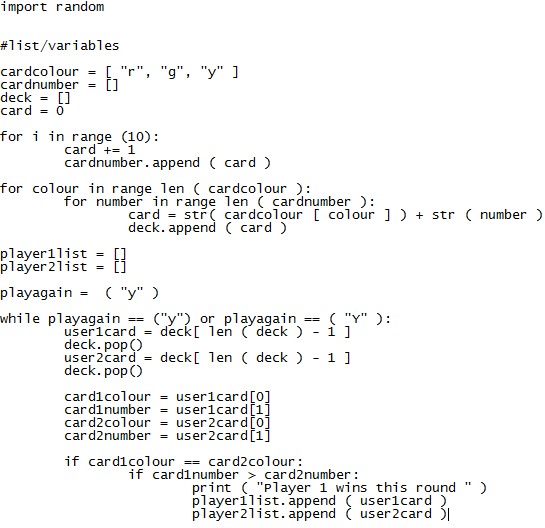




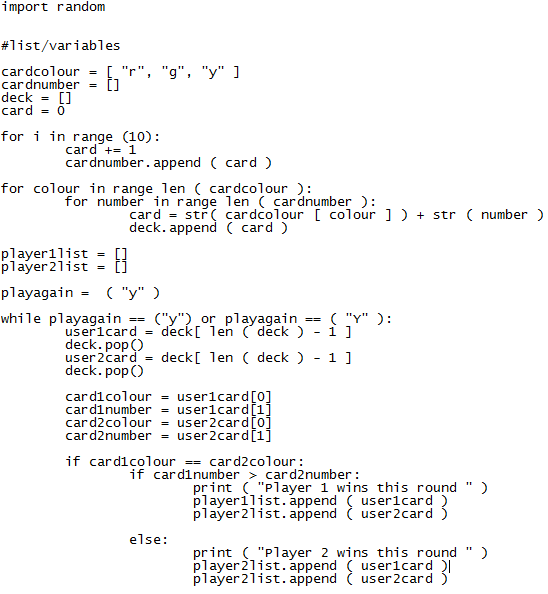
The next step is to split the cards into two separate variables that can be more easily compared. I do this by creating a variable that will store a single digit of the card, this is easy to do with my code because the colours are represented by a single letter rather than a full word. The first digit is the colour and the number is the second digit, however, Python counts from 0 so the first digit is variable[0]. In this examples the variables I want to split are user 1’s card and user 2’s card.



Next I need to compare the cards. I begin by checking if the cards’ colours are the same.

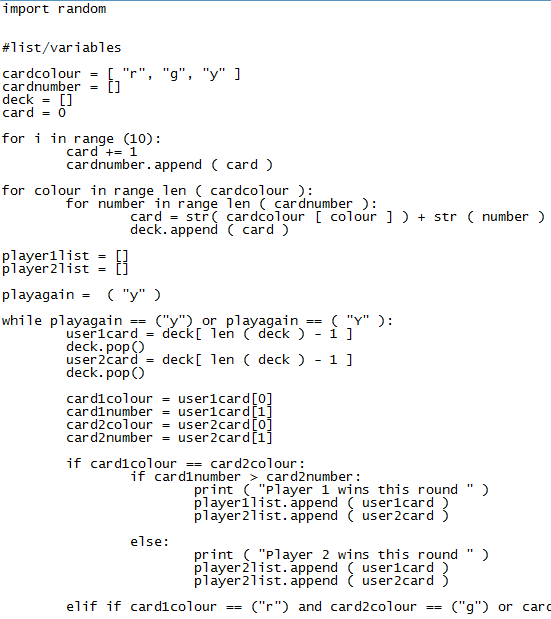


If the card’s colours are equal then I check if card 1 has a greater number than card 2, if it does then the code tells the player who won the round and gives both cards to player 1.

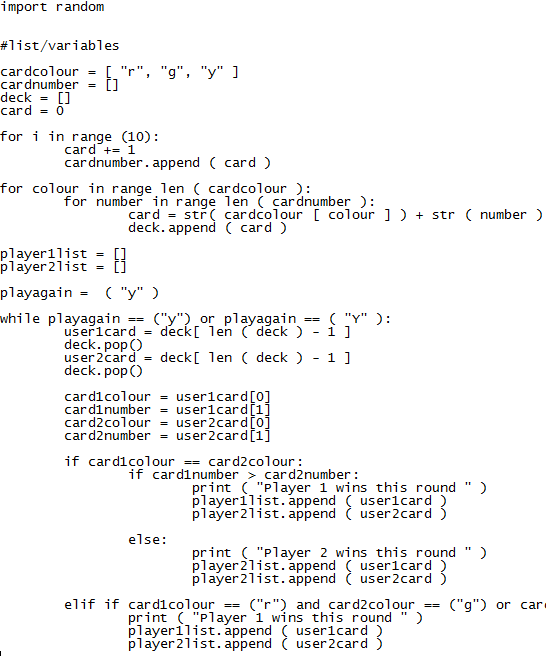


If player 1’s card hasn’t got a larger number then there can only be one other option, player 2 has a larger card number. Because of this I don’t need to compare the numbers a second time and only need to use an else command. This time around, the system tells the players that player 2 won and gives both of the cards to player 2.

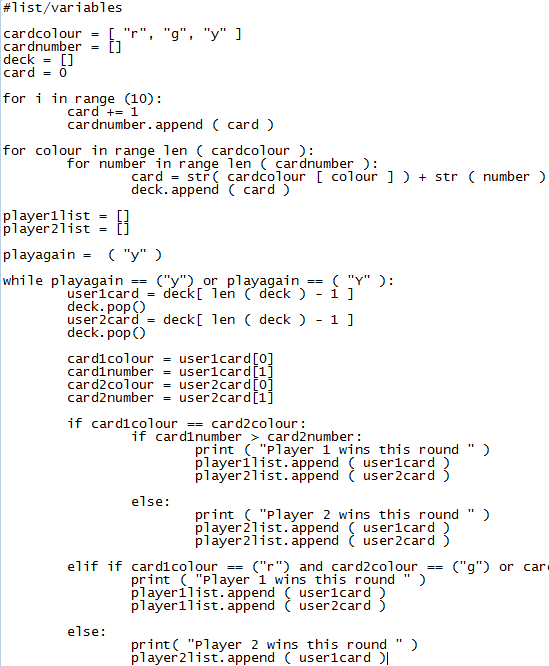




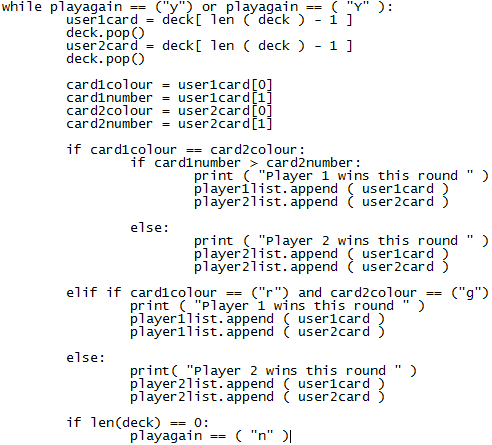
Next I have to compare the colours. At the time I was writing this code, this horrible line is the only method I could think of how to do this (I now realise that there is probably a far more efficient method). What it does is it individually goes through each colour combination where player 1 wins, so red and green, green and yellow, or yellow and red.



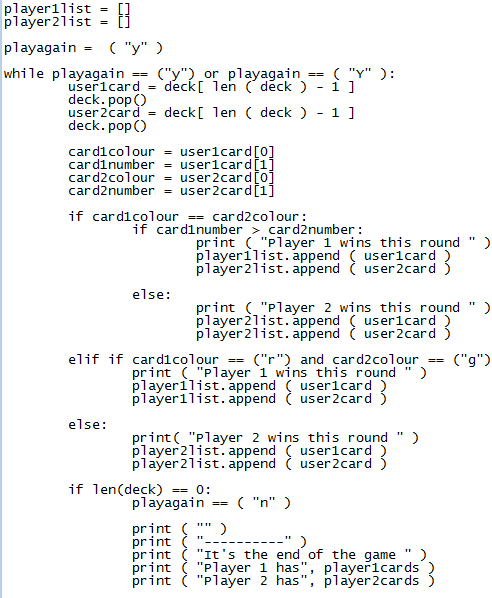
If the players’ cards fit into one of the pairs where player 1 wins then the system tells the players that player 1 wins and gives player 1 both of the cards.



If player 1 hasn’t won of course the only other player who can win is player 2 so if player 1 doesn’t win then player 2 gets the cards and congratulations. Luckily, this means that I can just use an else command.

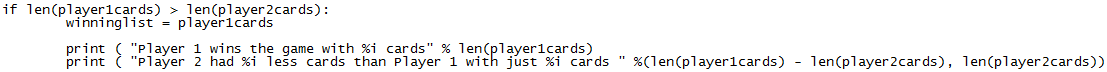


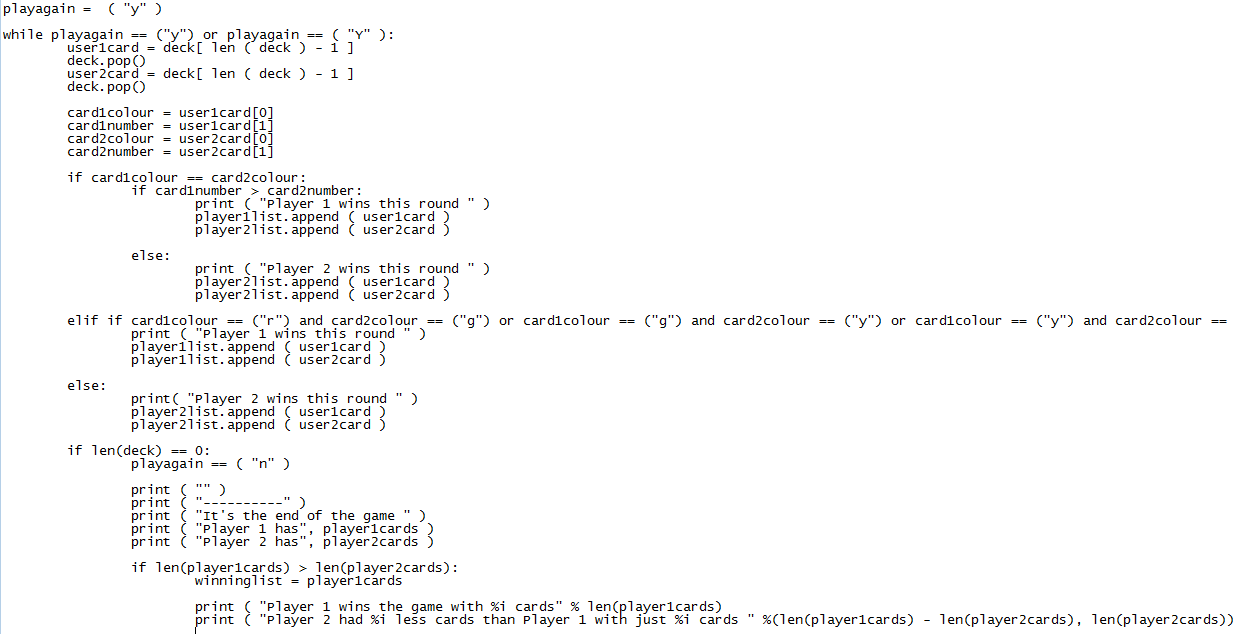
Next I had to make sure that the programme wouldn’t try to run after the cards have run out. This can be done with a simple IF loop. If the deck is empty, I set the PLAYAGAIN variable to ‘N’ so that the code does not loop again and the code will no longer prompt the user asking if they want to play again.



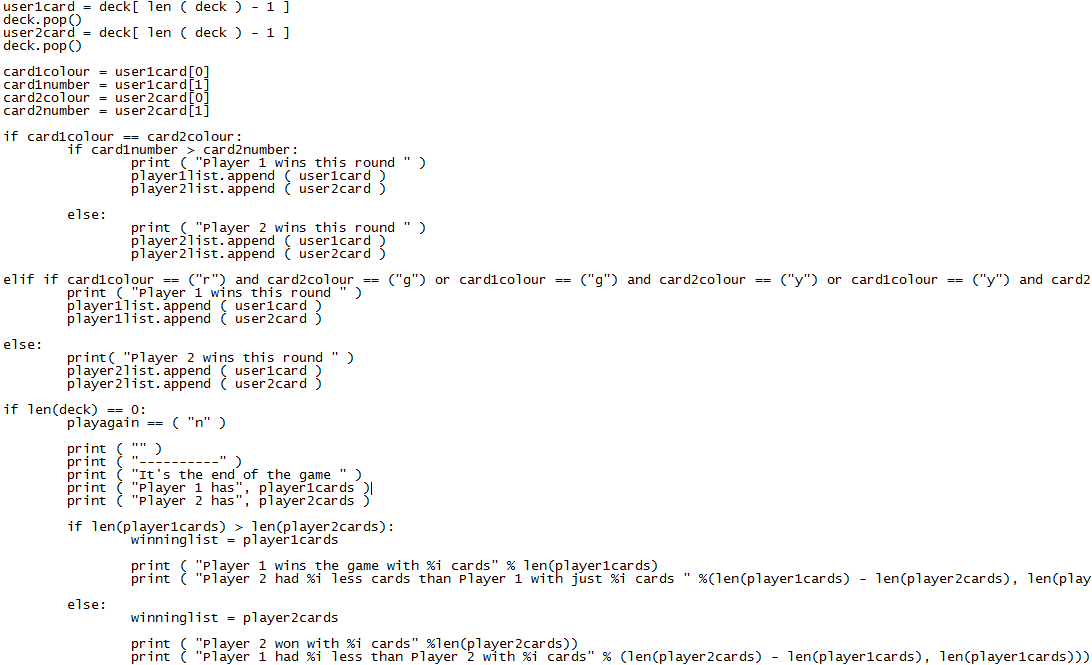
After the game has ended, I want to make sure that the user knows so I add a gap and print a little line to make it clear that what is going on is important. The code then announces the end of the game and shows the players who has what cards.

Normally I would use a % insert in this version of python however I don’t know how to do that with a list. If I were using Python 3.6.0 I’d use an FSTRING.

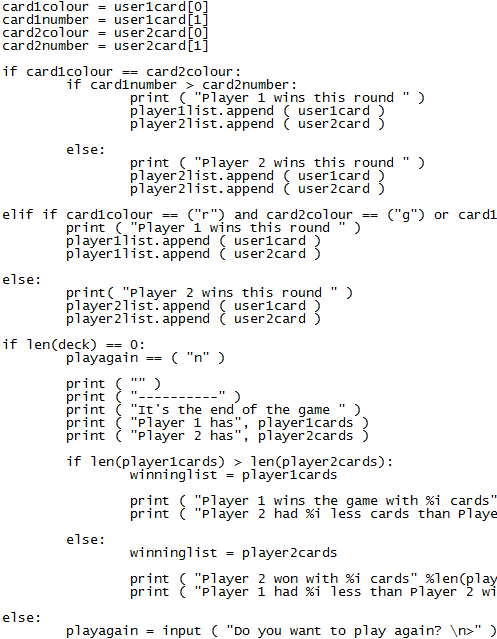




The next part I had to do was to prepare the winner’s list for being saved to a text file, to do this I need to set the variable WINNINGLIST to be the list of the player that won and then import that variable to an external script that will save it to a text file. This code also displays the winner and the difference in the amount of cards that the two players have.

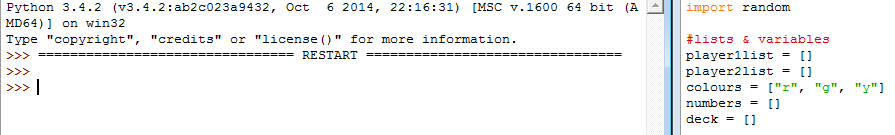


The same thing happens here except the winner is Player 2

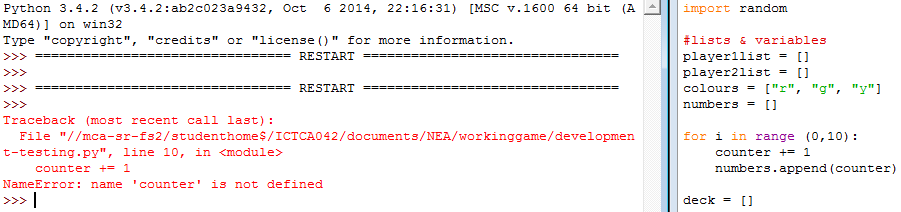


This code runs if there are cards in the deck and it simply ask if the players want to keep going and stores the response in the variable PLAYAGAIN.

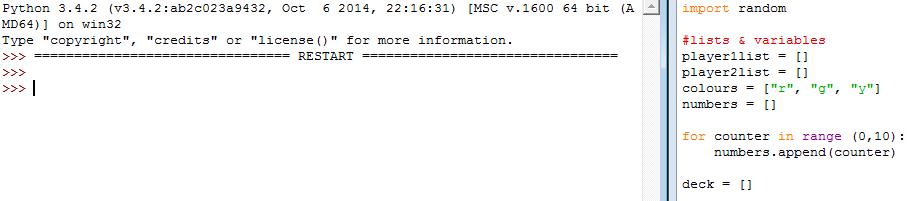
Testing



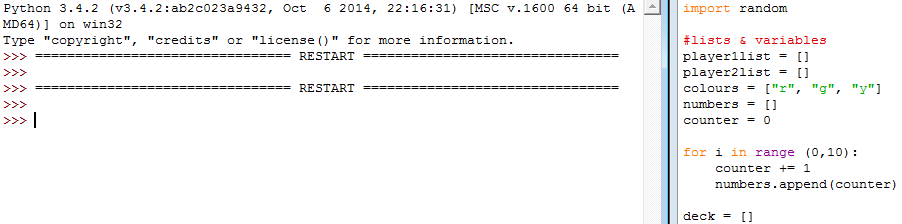
Here I was testing that Python was happy with all of the list names so that there would be no errors.



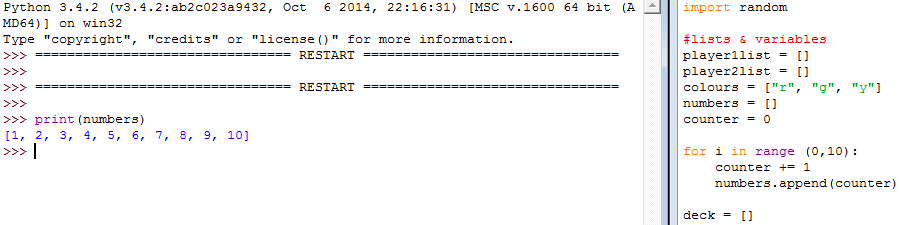
Python requires that variables are defined before operation can be performed on them, this is a simple fix.



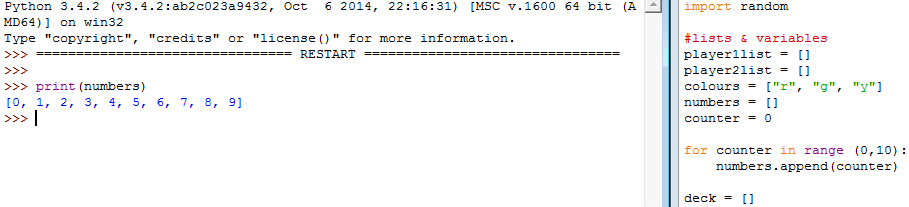
I don’t need to define a separate variable because the variable that keeps a track of how many times the loop has run would do the same thing as this:



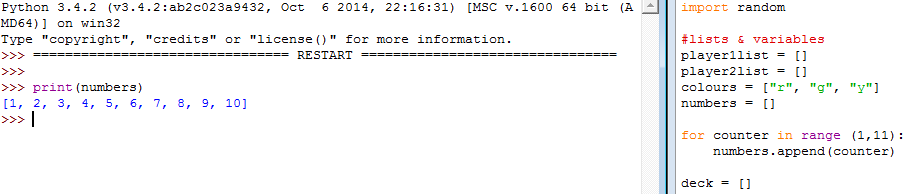
Now I need to make sure that the loop has in fact done its job correctly.



The inferior option works perfectly, however it’s still 2 lines clunkier.



Uh-oh, that’s not what I want however I do know what has gone wrong here.



I counted from 0 to 10, instead I need to do 1 to 11. This option is less expensive lines-wise, and just as easy to read (it’s more Pythonic).