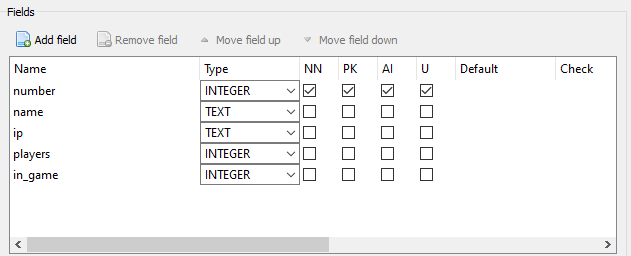
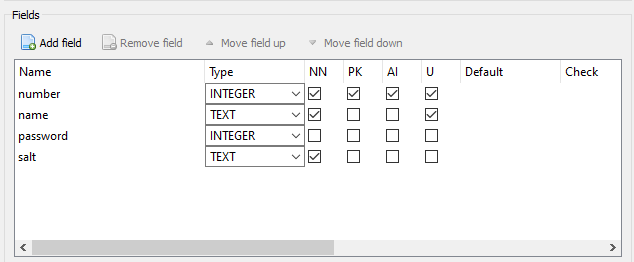
# development section

## server-side database creation

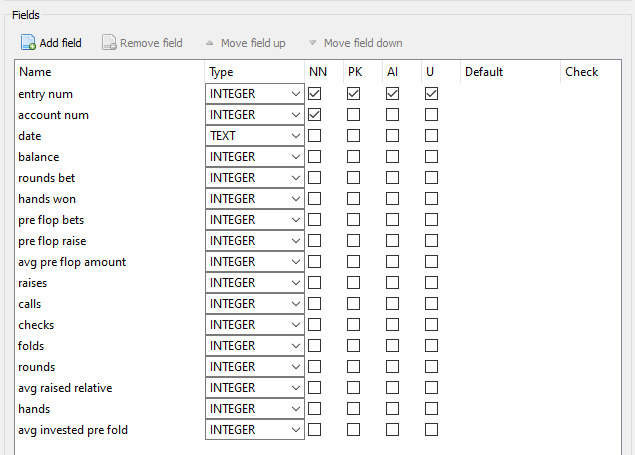
### Server list table creation and settings.



### Accounts table creation and settings.



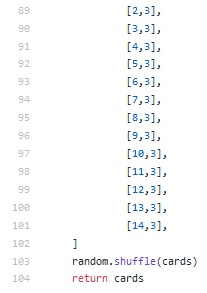
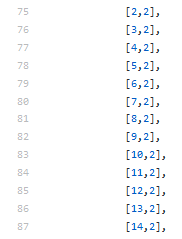
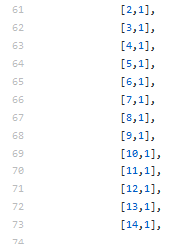
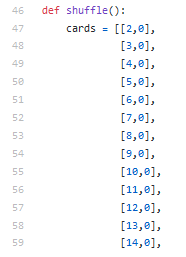
### Statistics table creation and settings.



## create the deck

### First, I made a prototype of the poker game which worked only on one device and was full command line to rapidly test and iterate on the mechanics to make sure it all works as intended.

### Creating the deck of cards, the shuffle function.



#### Since the design I made aces high as it makes it leads to fewer edge cases and exceptions

## the hand checking functions

### The check straight function.

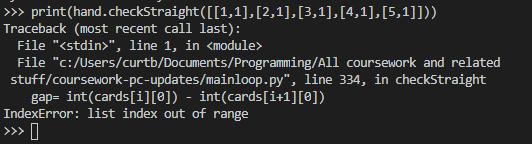
#### Code version 1

#### 

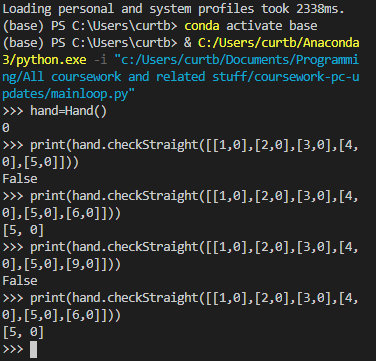
#### Test results

#### 

Dataset was not a straight so the function returns false.



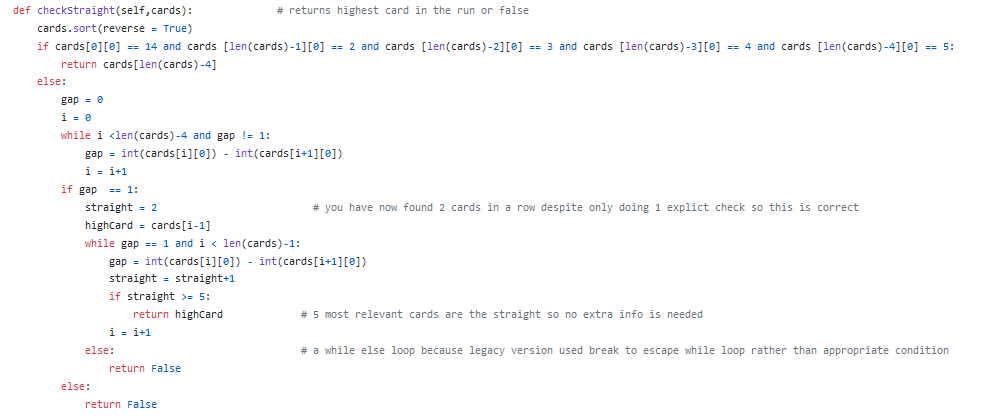
Dataset a straight but the function errored



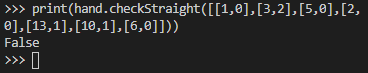
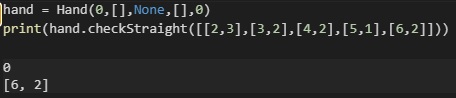
Even once the error was fixed it now needed straights to be runs of 6 rather than the intended 5

The code also has no handling for aces being high numbers which needed to be added after the change was made from the initial design.

#### Revised code



#### Re-testing new solution



returns the highest value from the straight as intended.

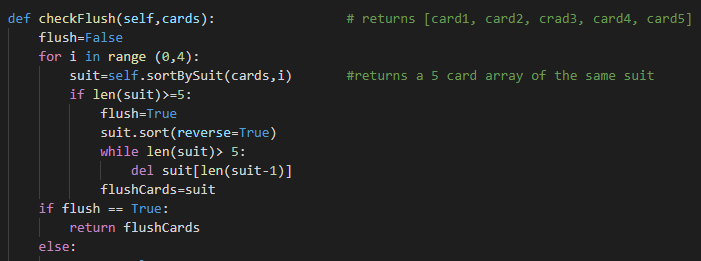
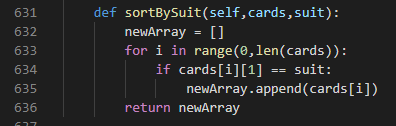
Dataset was not a straight so the function still returns false.

#### Evaluation

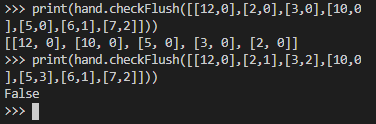
The code now works and returns the highest value from the straight which is what can be used to compare to other straights later in the development.

### The check flush function

#### Code



#### Tests

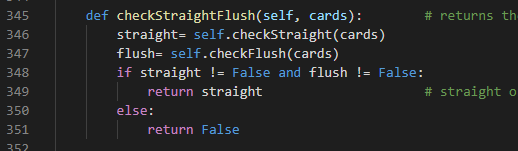


#### Evaluation

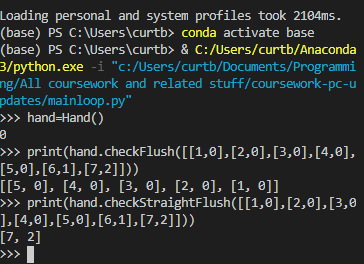
The code returns the full 5 cards if there is a flush which is important as comparing 2 flushes could go down to the lowest card if the four highest are identical. The code returned false on non-flush hands which is intended

### The check straight flush function

#### Code version 1

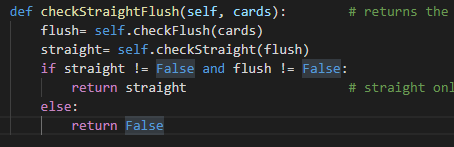


#### Test 1

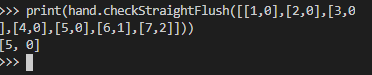


Here a flush and a straight made up of 2 different sets of 5 cards is registered as a straight flush which shouldn’t happen.

#### Code version 2



#### Test 2

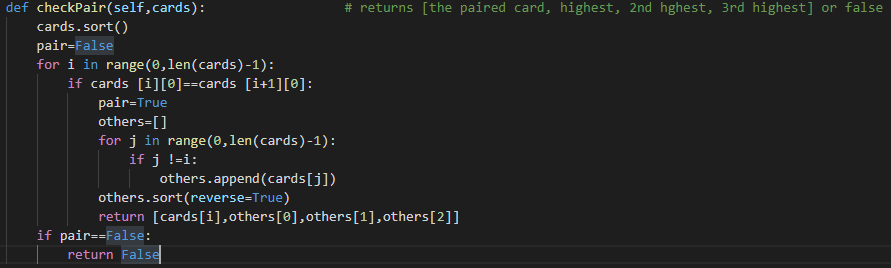


#### Evaluation

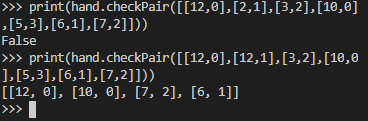
The code now performs as desired by only passing the 5 flush cards into the straight.

### Check pair function

#### Code



#### Testing

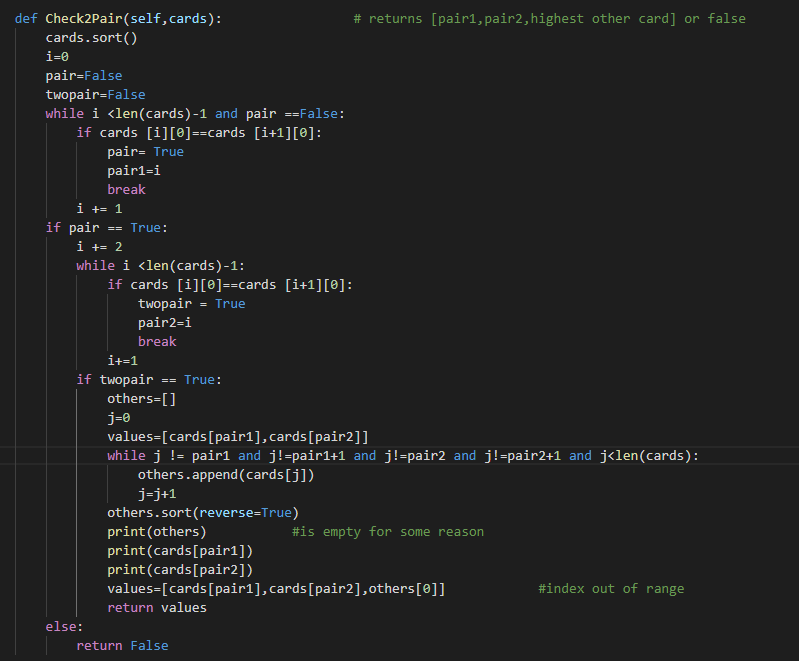


#### Evaluation

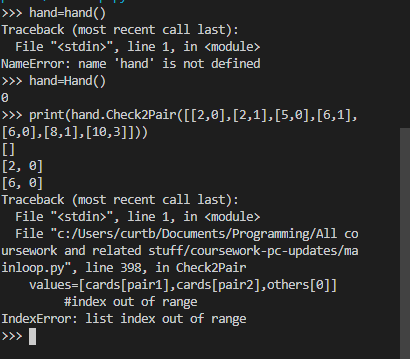
The code returns the 3 non paired cards as well as one of the pairs so if people have opposing pairs the comparison can go down to the third kicker if necessary.

### The check 2 pair function

#### Code version 1



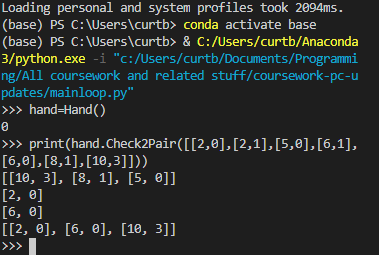
#### Test 1



#### Code version 2



#### Test 2

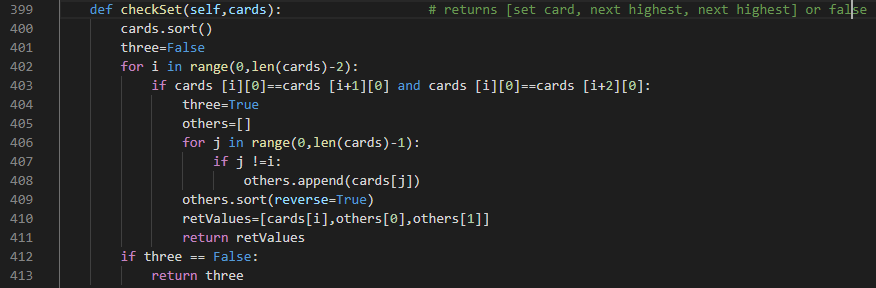


#### Evaluation

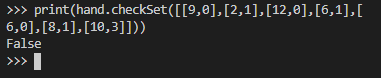
It returns the pairs and the kicker, using the while loop with poorly designed exit clauses lead to index out of range issues, using a for loop allowed for greater robustness and better exit clauses.

### Check Set function

#### Code



#### Tests

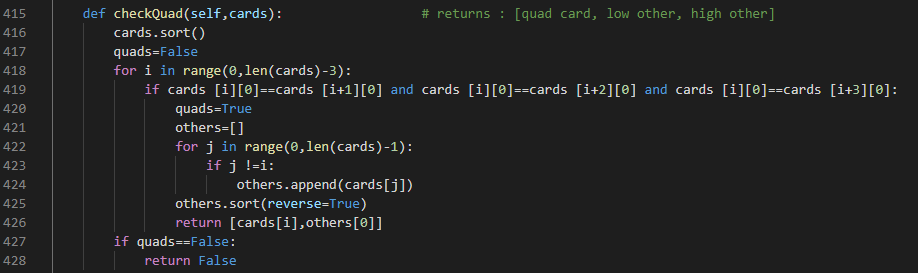


#### Evaluation

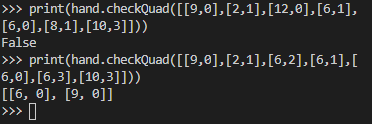
It performs as expected however with retrospect the kickers are.t necessary as even if 2 people have a set they can’t both have an equal set meaning final comparisons however returning these values won’t break anything or affect the end user in any meaningful way.

### The Check quad function

#### Code



#### Tests



#### Evaluation

It performs as expected however with retrospect the kickers are.t necessary as even if 2 people have four of a kind, they can’t both have an equal card meaning final comparisons however returning these values won’t break anything or affect the end user in any meaningful way.

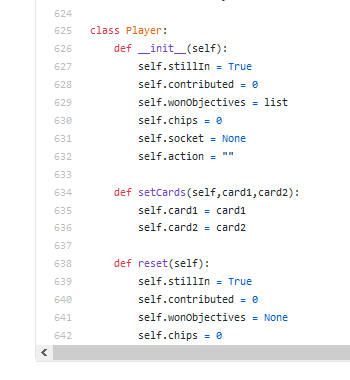
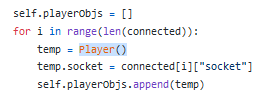
## Switching from dictionary to custom objects

Early in development I used a dictionary which was keyed using player numbers making it functionally no different to an array, each item then had an array which contained useful information, this array was ambiguous and hard to actually understand and should have been dictionary or a custom object. This was a change I then made replacing the original system with an array of player objects.

### Dictionary format



### Player class

## comparing the hands

### After creating all of the functions to compare the hands I then needed a function which could use the returned values and evaluate the winners of the hand, it needed to be able to deal with any number of players folded or not and give an output that would allow another function to then use it to allocate the chips to the players. The function comprises of 2 major loops, the first finds out what you have in your hand using the above functions and the second compares these and ranks the hands

### Part one, evaluating the hands

#### The code

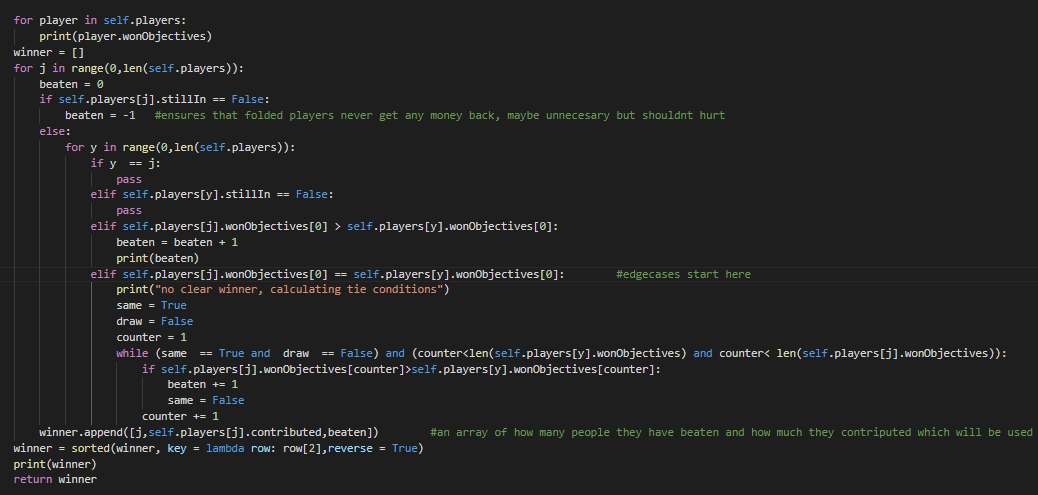


#### Evaluation

This segment calls the functions on the players cards + the centre cards in descending order meaning it will only recognise the most valuable attribute, this is then assigned to the eon objectives property of the player and used in the next loop.

### Comparing the hands

#### The code



#### Evaluation

This section of the code comprises primarily of a nested for loop iterating over the players array and finding who has the highest number. If the numbers are the same it goes through the kicker cards and comparing it against each other. This array is then sorted into reverse order so that the allocate chips function can use it to allocate winnings accordingly, more information than just the winner is needed in case of split points were the winner in ineligible for a portion of the bid chips.