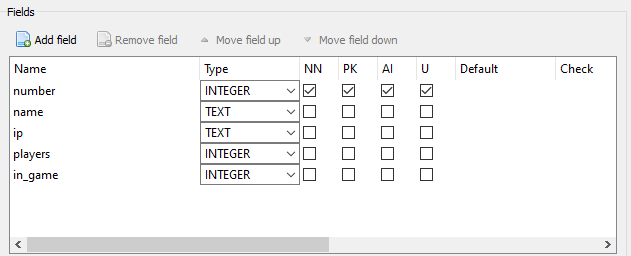
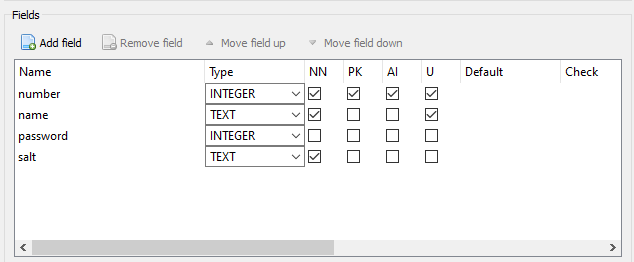
# development section

## server-side database creation

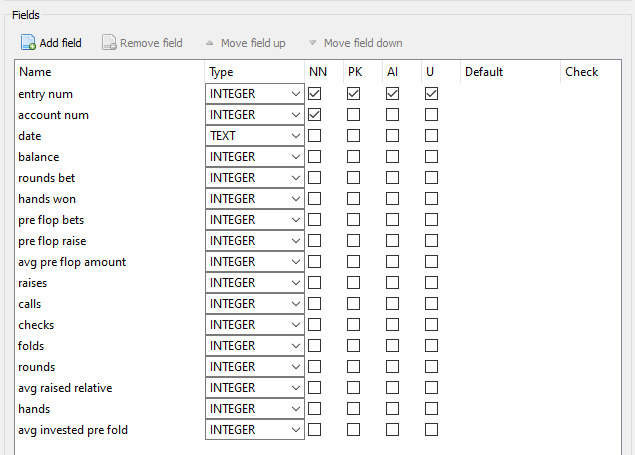
### Server list table creation and settings.



### Accounts table creation and settings.



### Statistics table creation and settings.

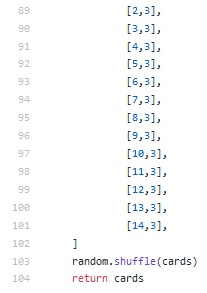
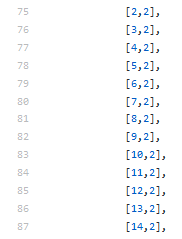
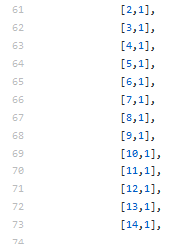
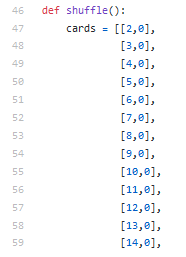


## The command line prototype

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.

## dealing the cards

### 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 flop river and turn functions

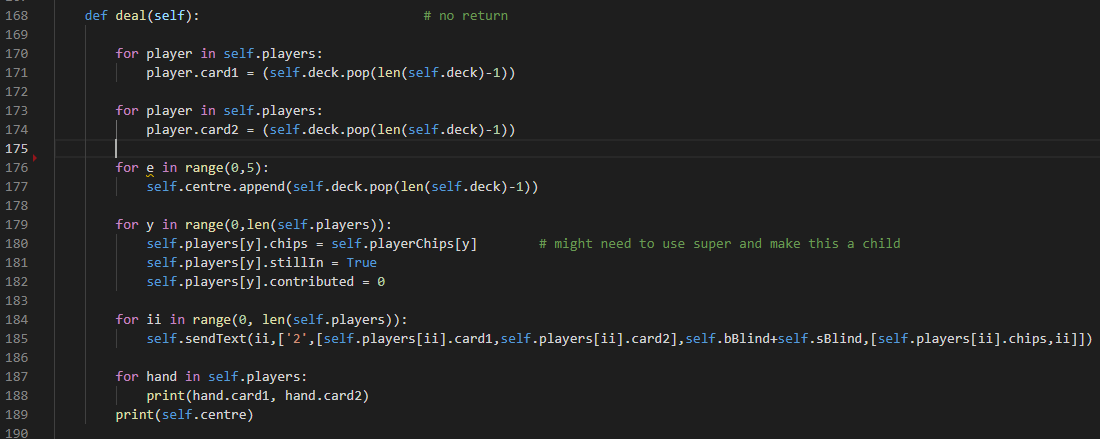
#### The code

#### Evaluation

This code selects the appropriate cards from the centre, combines them with the tag which the user client uses to establish which labels to change. The card is returned but this isn’t used in the final version of the application.

### The deal function.

#### The code



#### Evaluation

This function is essentially a constructor which populates the player objects in the players array. It also handles sending information to the players that they will need at the start of each hand.

## the hand checking functions

### The check straight function.

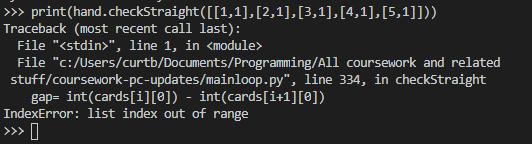
#### Code version 1

#### 

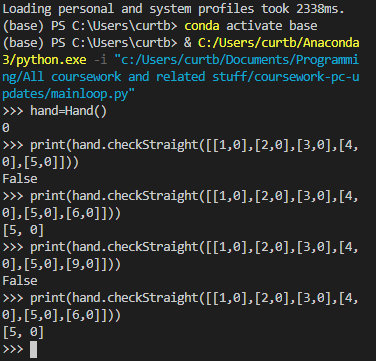
#### Test results

#### 

Dataset was not a straight so the function returns false.



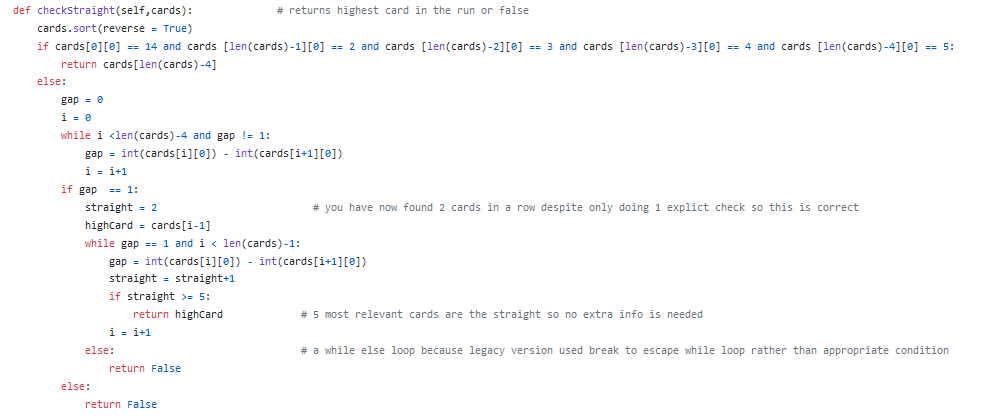
Dataset a straight but the function errored



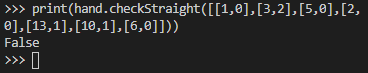
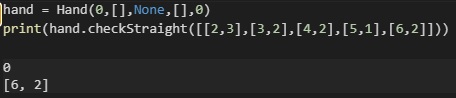
Even once the error was fixed it now needed straights to be runs of 6 rather than the intended 5

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.

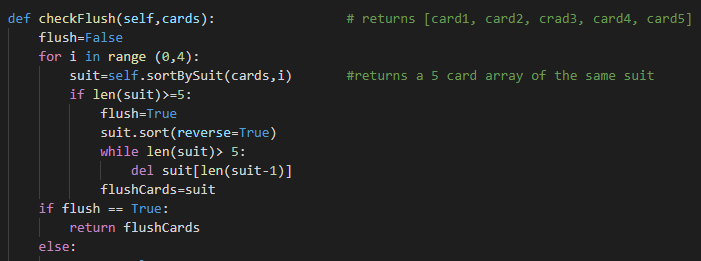
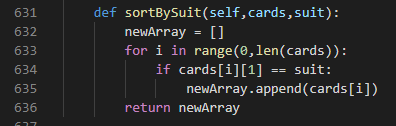
Dataset was not a straight so the function still returns false.

#### Evaluation

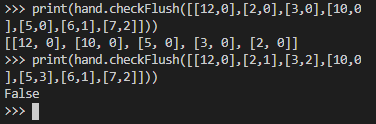
The code now works and returns the highest value from the straight which is what can be used to compare to other straights later in the development.

### The check flush function

#### Code



#### Tests

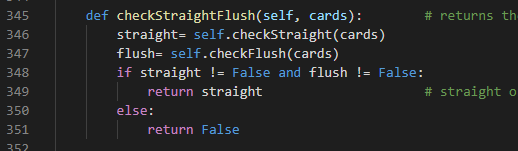


#### Evaluation

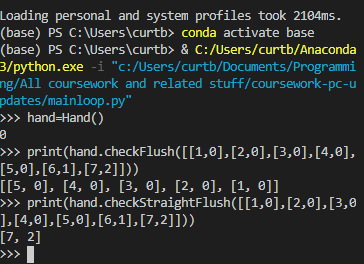
The code returns the full 5 cards if there is a flush which is important as comparing 2 flushes could go down to the lowest card if the four highest are identical. The code returned false on non-flush hands which is intended

### The check straight flush function

#### Code version 1

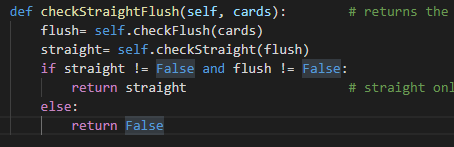


#### Test 1

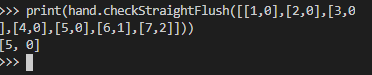


Here a flush and a straight made up of 2 different sets of 5 cards is registered as a straight flush which shouldn’t happen.

#### Code version 2



#### Test 2

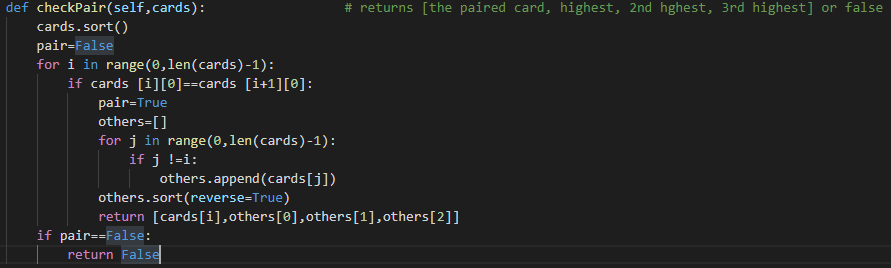


#### Evaluation

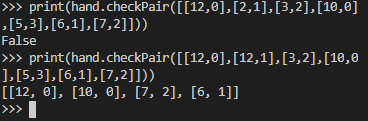
The code now performs as desired by only passing the 5 flush cards into the straight.

### Check pair function

#### Code



#### Testing

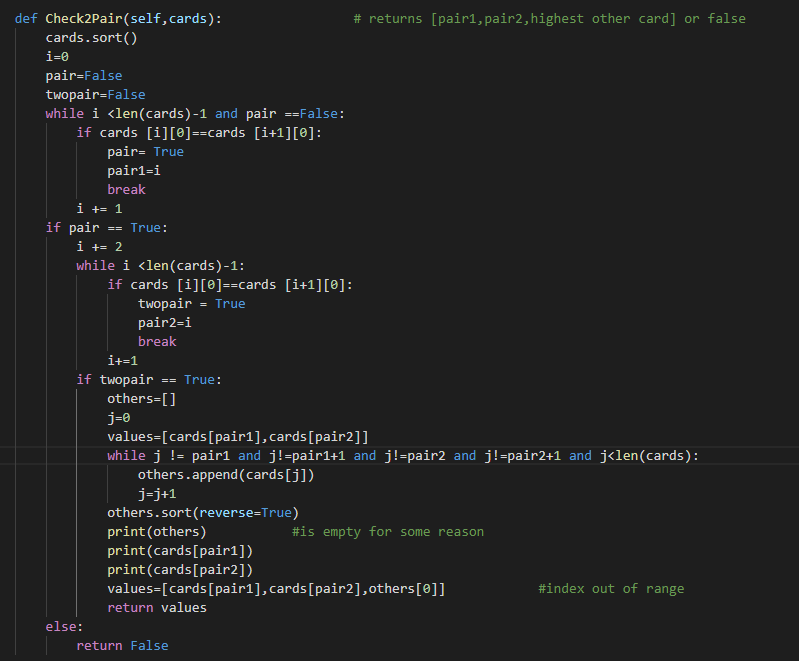


#### Evaluation

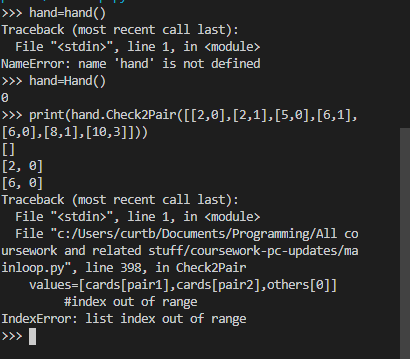
The code returns the 3 non paired cards as well as one of the pairs so if people have opposing pairs the comparison can go down to the third kicker if necessary.

### The check 2 pair function

#### Code version 1



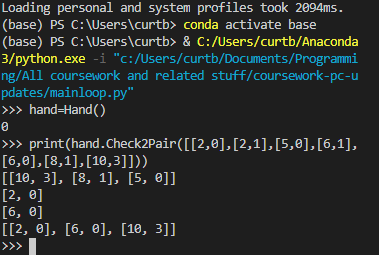
#### Test 1



#### Code version 2



#### Test 2

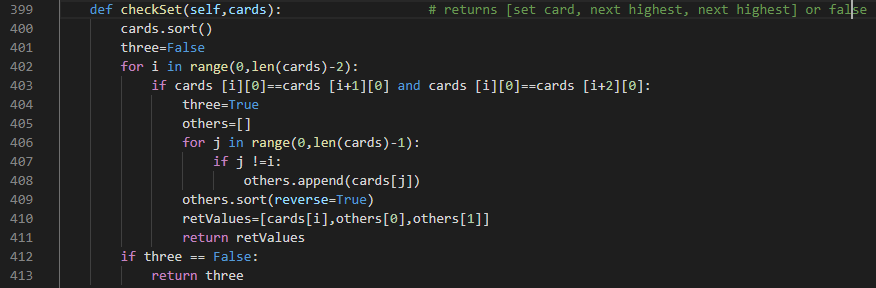


#### Evaluation

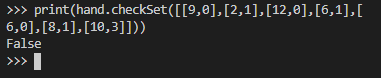
It returns the pairs and the kicker, using the while loop with poorly designed exit clauses lead to index out of range issues, using a for loop allowed for greater robustness and better exit clauses.

### Check Set function

#### Code



#### Tests

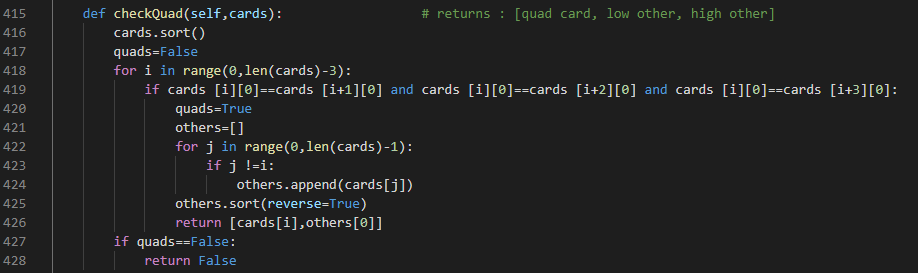


#### Evaluation

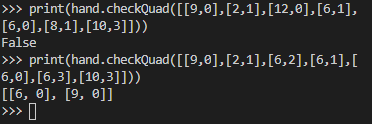
It performs as expected however with retrospect the kickers are.t necessary as even if 2 people have a set they can’t both have an equal set meaning final comparisons however returning these values won’t break anything or affect the end user in any meaningful way.

### 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.

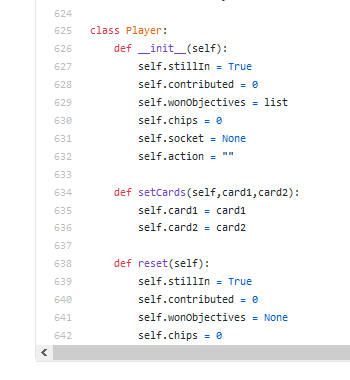
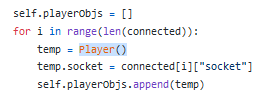
## Switching from dictionary to custom objects

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

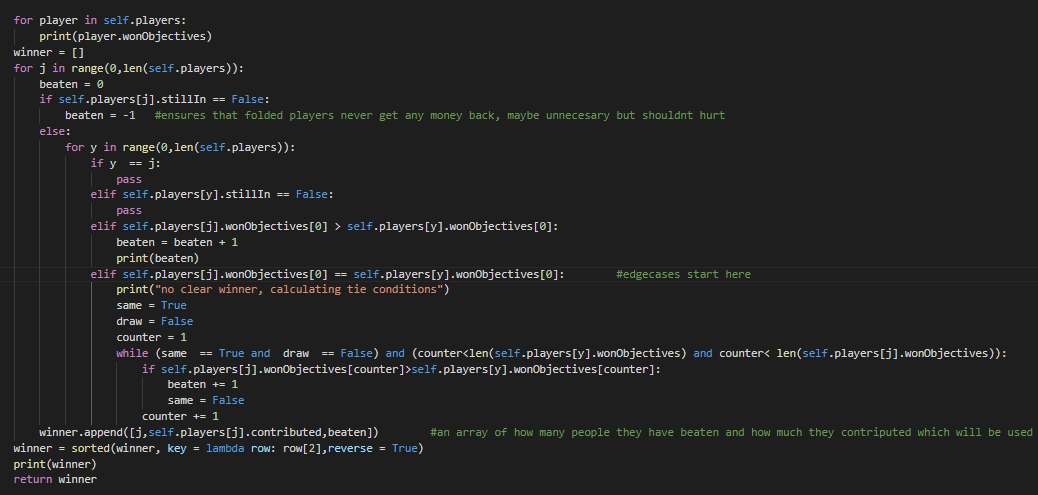


#### Evaluation

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

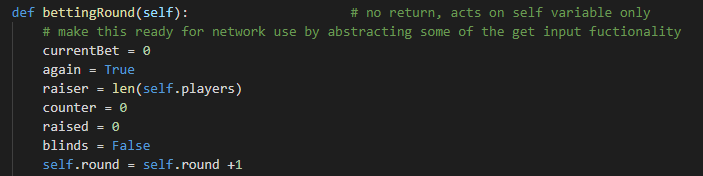
This section of the code comprises primarily of a nested for loop iterating over the players array and finding who has the highest number. If the numbers are the same it goes through the kicker cards and comparing it against each other. This array is then sorted into reverse order so that the allocate 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.

## Betting rounds

### Explanation

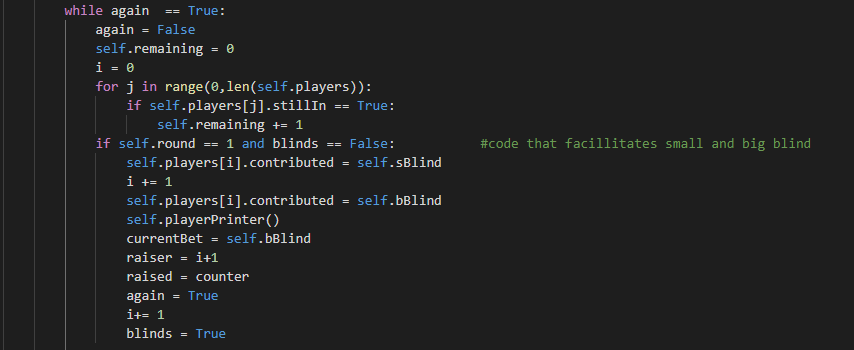
The betting round function processes all of the inputs needed for the handling of peoples actions during the game. It is the most complex function of the game loop itself and works on properties that have essentially been passed by reference rather than on local variable which it then returns. This allows for it to make changes to multiple properties and means any function it calls during execution have the most up to date values for those properties.

### The code – part 1



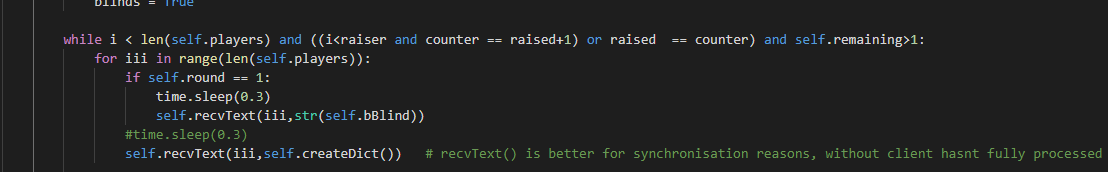
The first section of the function creates the local variables needed at gives them their starting values as well as incrementing which round the game is on by one.

### The code – part 2



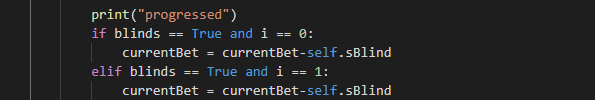
The rest of the function takes using a nested while loop, one may only be ran once but is used if someone raises and therefor you need to go round the table for more people to call. The part pictured here deals with the blinds and is therefor only ran if it is round one and the blinds have not yet been taken.

### The code – part 3



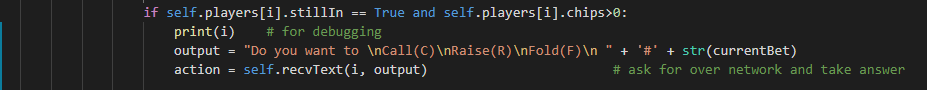
This is the start of the second while loop which loops until it has handled every player or not gotten round to whoever raised that round or until everyone but one person has folded. Here I had to use my send and receive function as I was getting issues with transmissions not being received by the client so requiring a confirmation helps, as does adding a brief sleep period otherwise the client is still processing previous requests before this one arrives. The full networking system will be addressed later.

### The code – part 4



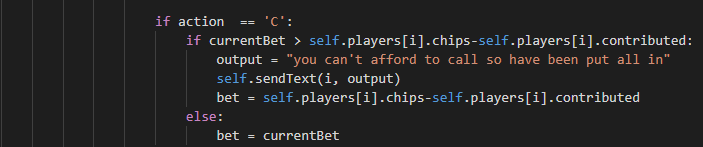
This section actually subtracts the earlier calculated blinds from the first and second player.

### The code – part 5



This is the main section of the function that begins by checking the current player is eligible to make an action by checking if they are folded and if they have any chips to bet. It then takes an input from the user, the actual test being sent in never seen in this version but was used both in the single player and multiplayer command line versions. This return value is then one of 3 things, C, F, or R standing for call, fold and raise each of these then have their own processing else/if statement.

### The code – part 6



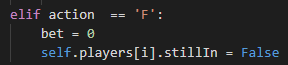
This section processes what happens if a user wants to check or call by adding the current bet to the players bet local variable which once the loop has completed will be added to the player’s contributed attribute. It also checks if the player can afford it and if not notifies them and puts them all in.

### The code – part 7



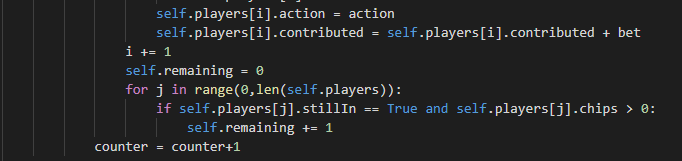
This part of the code handles the raising of bets, after receiving the decision to raise it then needs to ask by how much along with the tag 6 which tells the client that they need to open the raise ui. It also has several checks for if they raise by too much or can’t afford it etc. Many of these became irrelevant as the user interface doesn’t let you enter illegal values but keeping them for insurance in case the ui has an issue is a good idea.

### The code – part 8



This is a short part that handles folding, it has no checks as a player can fold at any time.

### The code – part 9



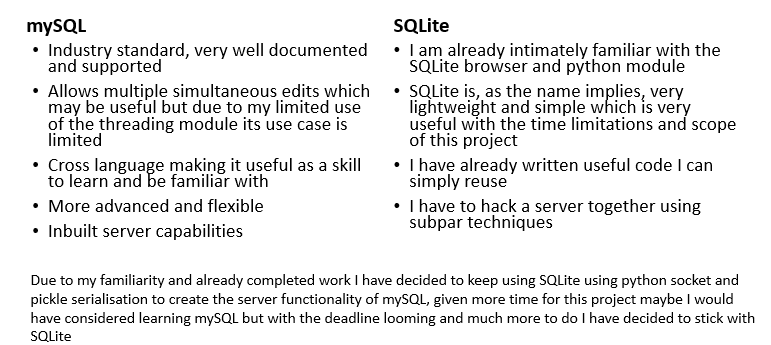
This is the final section of the loop which updates the attributes to reflect the changes of the local variables throughout the function. It also increments the local counters

## networking

### design philosophy and evolution of ideas

The networking in this project is integral to its function and therefor it was important I got it right. My initial plan was a client server model for statistics and choosing a game with a hole punched peer to peer network for the game itself. Unfortunately trying to hole punch for a variable number of clients with the time constraints of this project with the weight of the pandemic I choose to go with a pure client server network. This comes with the advantage that if you use the direct connect function even once my support for the project ends people can still play the game (as long as before ending support I remove the accounts system).

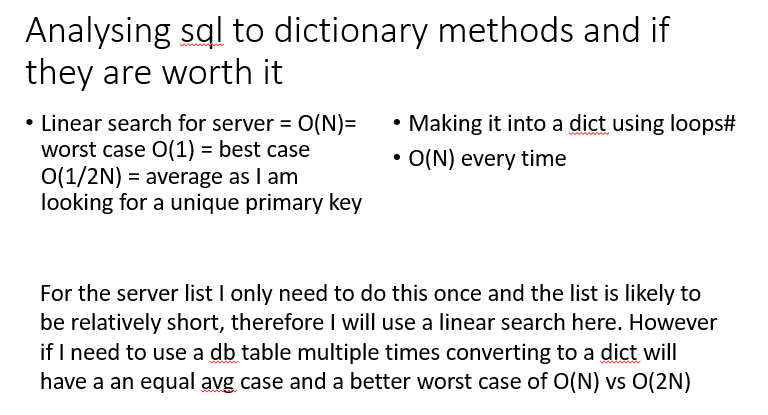
For the database retrieval I choose to make my own networking system using python sockets and sqlite. This is not technically the best solution but my familiarity with these modules allowed me to stay within the time constraints of the project.



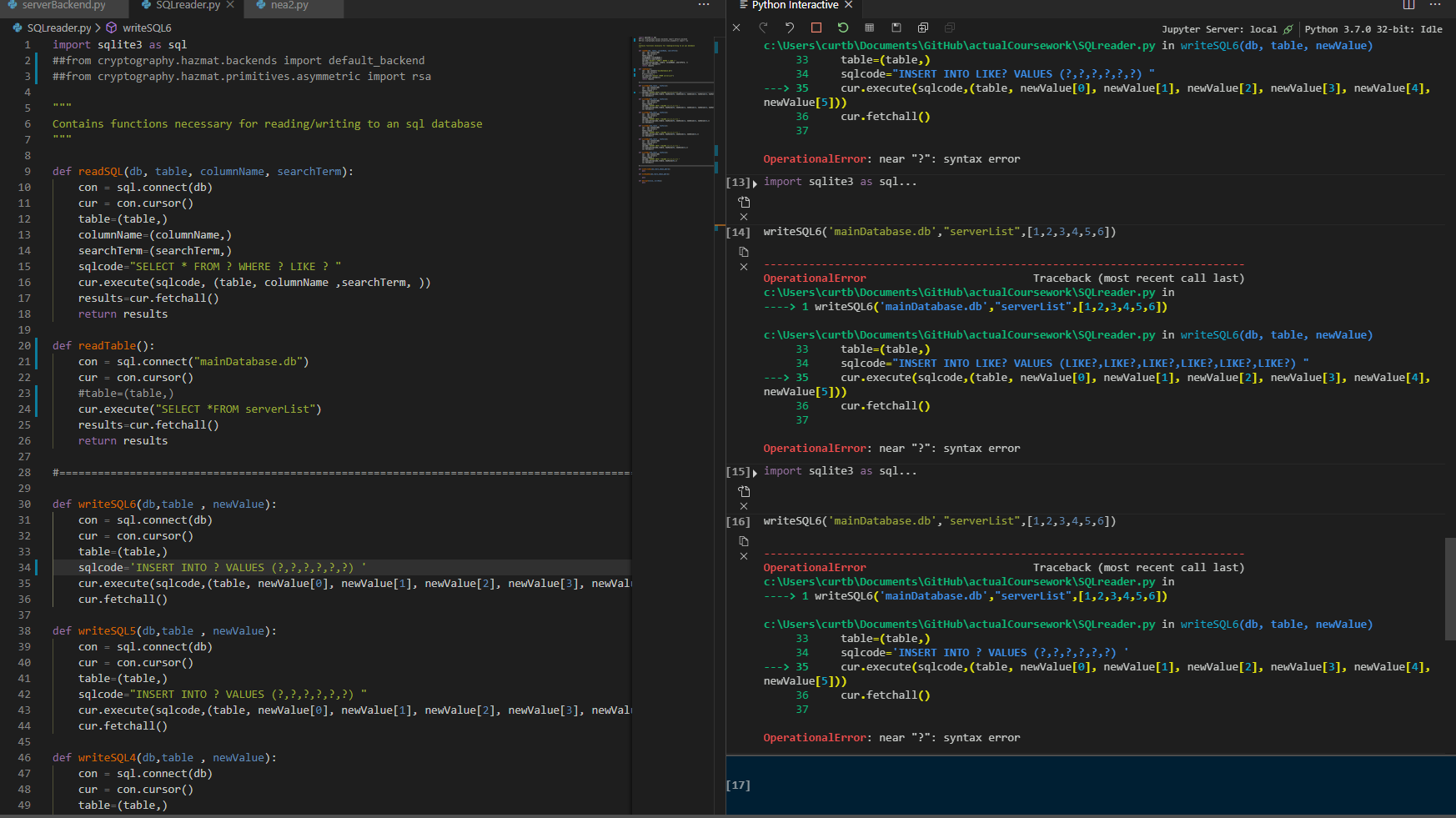
I started with basic prototype files that sent strings between 2 computers to build familiarity with the python socket api. Once this was done I began work on the sql database management system.

### Database manager

#### Dicts vs lists for sql query returns



#### Bug #1

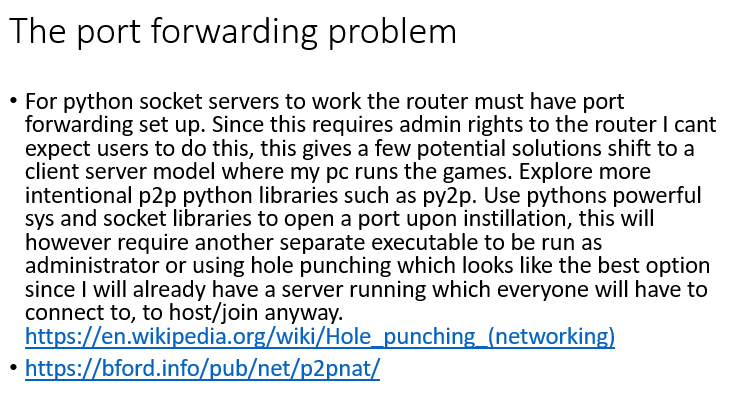


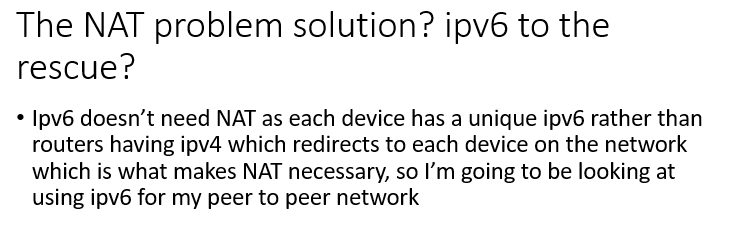
My initial plan was to use a general purpose custom library which can read or write to any record in any table taking all this information as parameters. Unfortunately sqlite does not allow the table name to be variable meaning each read and write needs its own function rather than a handful of general purpose ones. This also lead to me moving the cryptography to the main file using this for only reading and writing.

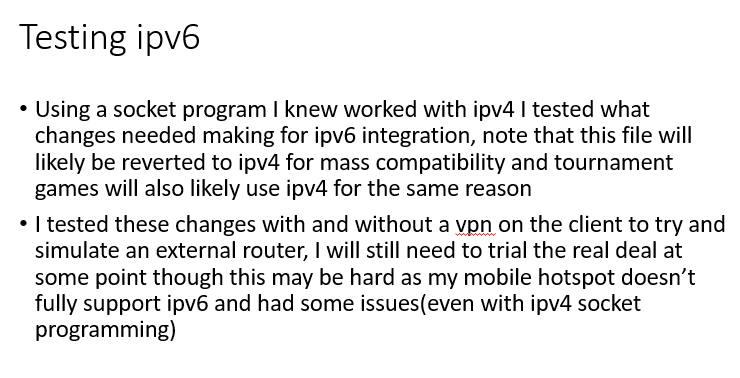
### Game hosting & peer to peer attempts

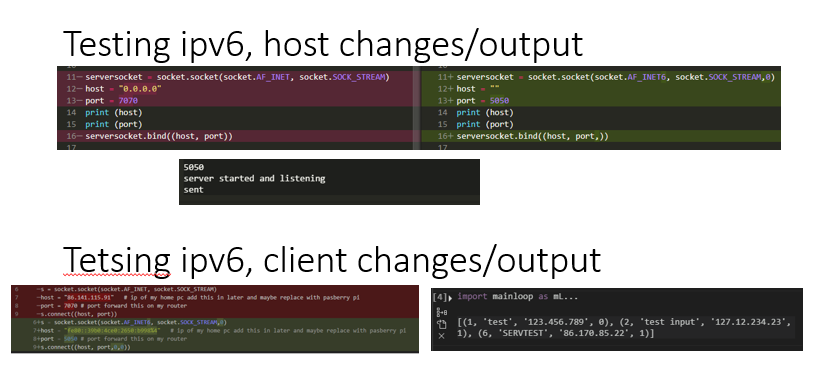
I originally wanted to pursue a peer to peer system as it would simplify peoples use of the product as if they wanted to host a server for their friends with little to no technical knowledge they could. However this Is a hard thing to do as even large online games such as Minecraft only have client server and do not support peer to peer networks so it is not a huge loss to the competition. Below is a map of my attempts to make peer to peer work using ipv6, custom libraries such as “ipv8” and a fully custom low level hole punching method based on this article all written while developing the program:

<https://bford.info/pub/net/p2pnat/>



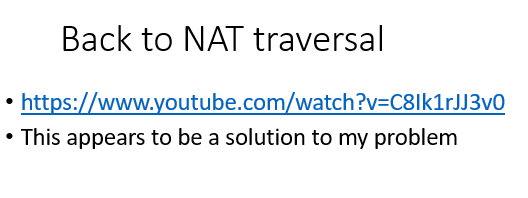


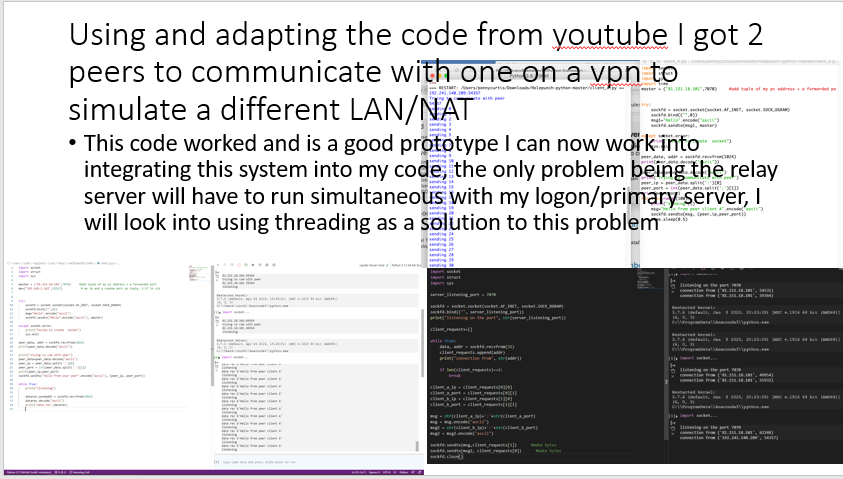
at this point in dev I still had big plans that never came to fruition and have now been moved to post release updates so I can meet the deadline



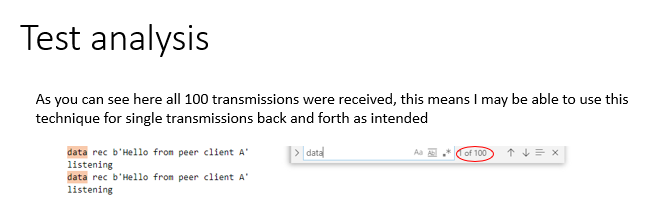
I was previously using link local ip not my public ip which is what caused issues whenoff of my lan.

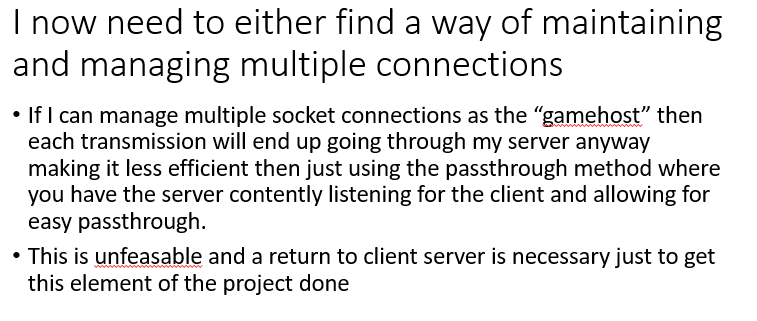
Note: this ipv6 solution was abandoned as it is not yet deployed sufficiently for me to feel comfortable using it for a solution which is accessible to anyone with a windows computer



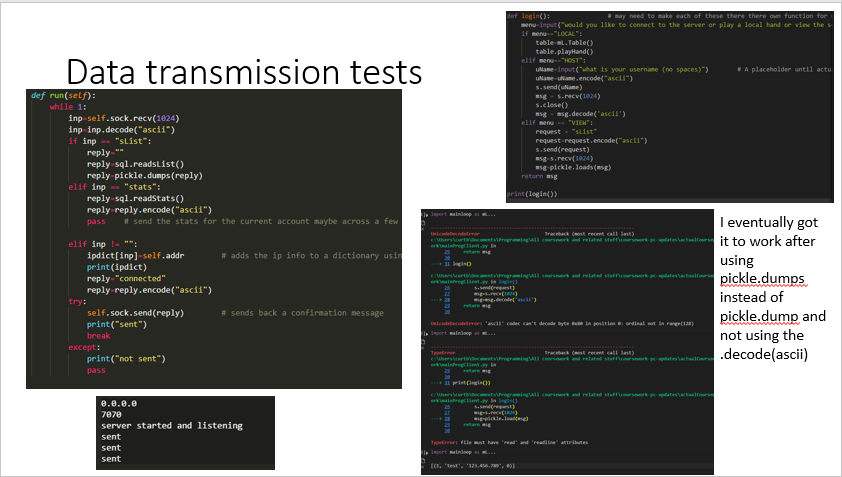


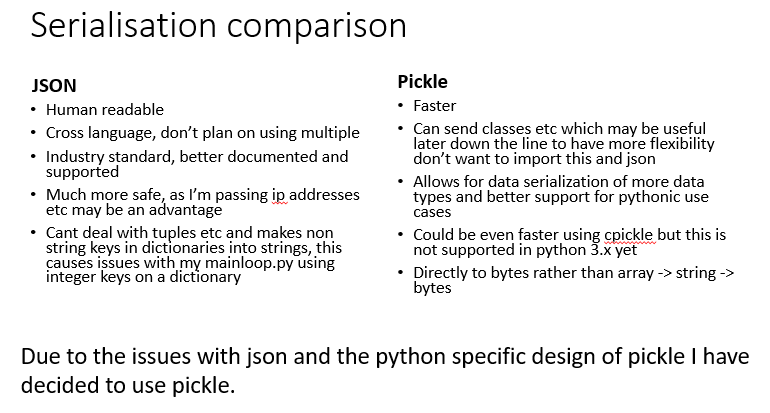
This was the closest I got to a successful hole punching system however adapting it for a variable number of clients proved unfeasable and the time invested into this part of the project was becoming larger than I had initially intended.





### Network communications code and testing



I switched from ascii encoding to pickle as both return bytes objects but by universally uning pickle it allows arrays, dictionaries and object instances to be sent not just strings. The alternative was a combo of ascii encoding and json

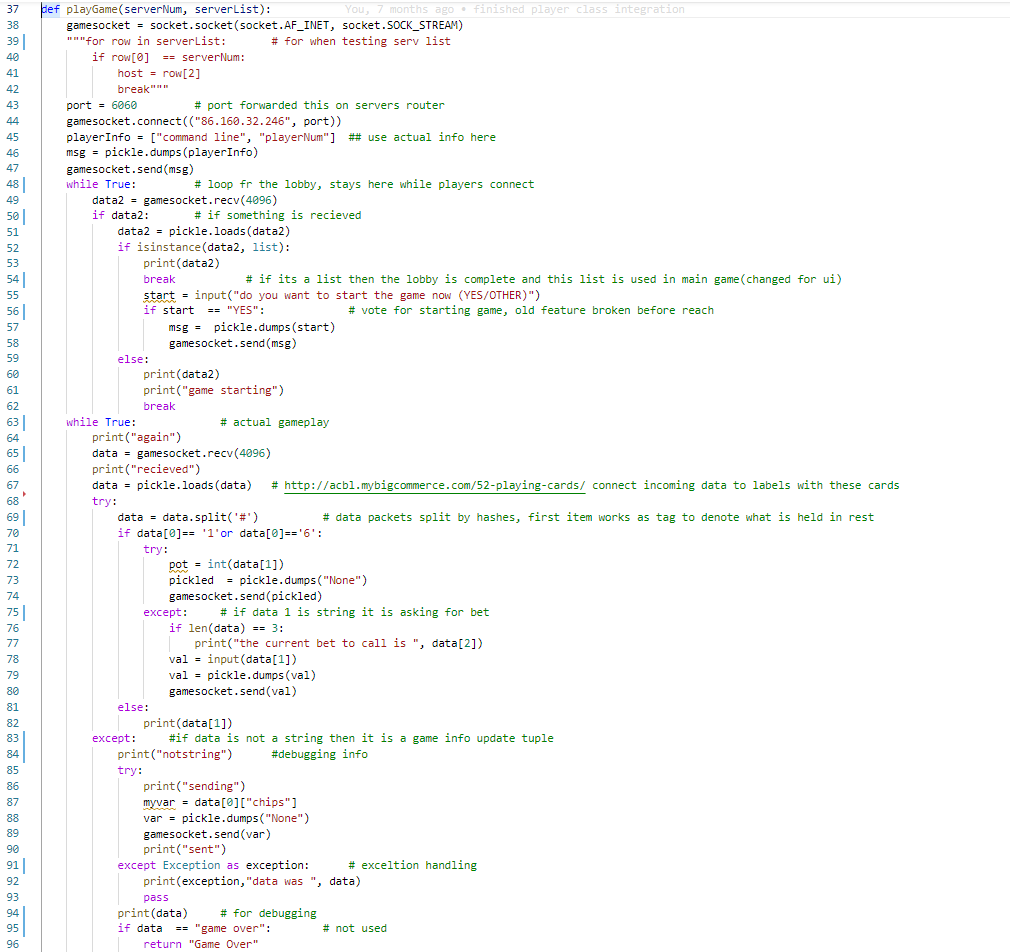
### User Interface and pre ui prototypes

I mostly stuck to my original ideas of a user interface with a menu server browser and game window all separated. I had initially planned to use qt designer but in the end I hard coded the ui as this allowed me to easily change things on the fly from within my text editor allowing me to rapidly prototype and removing the need for a .ui file which saves space and reduces complexity for instillation.

Initially I created a command line version which I could use to test networking protocols and functions before deploying them to the more complex and less changeable ui version. This file had functions for playing a game and used direct hard coded ip connection as it would never be used without me having the python file open anyway. The server browser was returned an array but this was purely to test the sql commands. This caused some issues as the ui and non ui version both had the same username meaning they clashed meaning the server didn’t know who to send data to. The code. Due to the large amounts of server side processing this code could be relatively short and was essentially ust a custom parser for the way I wrote my data packets. Its main function was to decipher tags and give me an easy way to see what data was actually beong received. This was then adapted into multiple separate functions in the final product

Issues:

The simplicity meant it could respond faster than the ui which meant it didn’t catch desynchronisation issues.



#### ui code – the constructor

The constructor creates the ui widgets and initialises them. I used Qframes to allow me to show and hide whole “pages” at the same time. I also used an unbound Qframe to frame the user login system and not have it part of the main window as I felt this was more professional. Obviously due to the use of code to do the ui over a .ui file this constructor is long and unreadable. In future if I wanted to make updates I would likely convert this into a .ui file to make it easier to work on and as I am unlikely to need the rapid prototyping which was the original reason not to use QTdesigners more immediately optimised format.

def \_\_init\_\_(self, parent=None):

self.netInfo = 0

QWidget.\_\_init\_\_(self, parent)

self.setGeometry(100, 100, 1280, 720)

self.chips = 0

self.players = None

self.username = "tool" # get from accounts system

self.potSize = 0

# Put the widgets here

self.opponentBox = QGroupBox()

self.startButton = QPushButton(self.tr("&Start"))

self.statsButton = QPushButton("statisitics")

self.optionsButton = QPushButton("settings")

self.printerLabel = QLabel("placeholder")

self.outputLabel = QLabel("placeholder")

self.selectionCRdo = QRadioButton("Call/check")

self.selectionRRdo = QRadioButton("Raise")

self.selectionFRdo = QRadioButton("Fold")

self.buttonConfirm = QPushButton("Enter")

self.flop1 = QLabel("card1")

self.flop2 = QLabel("card2")

self.flop3 = QLabel("card3")

self.flop4 = QLabel("card4")

self.flop5 = QLabel("card5")

self.hand1 = QLabel("hand1")

self.hand2 = QLabel("hand2")

self.potLabel = QLabel("0")

self.chipLabel = QLabel('0')

self.raiseGroup = QGroupBox()

self.raiseTxt = QSpinBox()

self.raiseTxt.setValue(0)

self.raiseSlider = QSlider(Qt.Horizontal)

self.raiseTxt.setMaximum(self.chips)

self.raiseSlider.setRange(0, self.chips)

self.raiseTxt.setSingleStep(1)

self.raiseSlider.setSingleStep(1)

self.raiseConfirm = QPushButton("Enter")

self.raiseLabel = QLabel("How much do you want to raise the bet by")

self.back = QPixmap("assetts/gray\_ba0k.png")

self.back3 = QPixmap("assetts/folded.png")

self.back = self.back.scaledToWidth(96)

self.back2 = self.back.scaledToWidth(48)

self.back3 = self.back3.scaledToWidth(48)

self.resetCards()

self.raiseLayout = QVBoxLayout()

self.subRaiseLayout = QHBoxLayout()

self.subRaiseLayout.addWidget(self.raiseTxt)

self.subRaiseLayout.addWidget(self.raiseConfirm)

self.raiseLayout.addWidget(self.raiseLabel)

self.raiseLayout.addWidget(self.raiseSlider)

self.raiseLayout.addLayout(self.subRaiseLayout)

self.raiseGroup.setLayout(self.raiseLayout)

self.raiseRow = QHBoxLayout()

self.raiseRow.addStretch(1)

self.raiseRow.addWidget(self.raiseGroup)

self.raiseRow.addStretch(1)

self.raiseGroup.hide()

centerLayout = QHBoxLayout()

centerRow = QHBoxLayout()

centerLayout.addWidget(self.flop1)

centerLayout.addWidget(self.flop2)

centerLayout.addWidget(self.flop3)

centerLayout.addWidget(self.flop4)

centerLayout.addWidget(self.flop5)

self.centerGroup = QGroupBox()

self.centerGroup.setLayout(centerLayout)

centerRow.addStretch(1)

centerRow.addWidget(self.centerGroup)

centerRow.addWidget(self.potLabel)

centerRow.addStretch(1)

handLayout = QHBoxLayout()

handrow = QHBoxLayout()

handLayout.addWidget(self.hand1)

handLayout.addWidget(self.hand2)

self.handGroup = QGroupBox()

self.handGroup.setLayout(handLayout)

handrow.addStretch(3)

handrow.addWidget(self.handGroup)

handrow.addWidget(self.chipLabel)

handrow.addStretch(3)

self.menuBrowserButton = QPushButton("Server List")

self.menuBrowserButton.clicked.connect(self.serverBrowser)

browserLayout = QVBoxLayout()

self.browserTable = QTableWidget()

self.browserButton = QPushButton("connect")

self.browserButton.clicked.connect(self.startGame)

browserLayout.addWidget(self.browserTable)

browserLayout.addWidget(self.browserButton)

self.browserFrame = QFrame()

self.browserFrame.setLayout(browserLayout)

self.browserFrame.hide()

self.radioGroup = QButtonGroup()

self.radioGroup.addButton(self.selectionCRdo)

self.radioGroup.addButton(self.selectionRRdo)

self.radioGroup.addButton(self.selectionFRdo)

self.radioGroup.addButton(self.buttonConfirm)

self.quitButton = QPushButton("Quit Game")

self.thread = Worker(self)

self.thread.finished.connect(self.threadDied)

self.startButton.clicked.connect(self.startListener)

self.loginFrame = QFrame()

self.usernameBox = QLineEdit()

self.pwordBox = QLineEdit()

self.pwordLabel = QLabel("Password")

self.usernameLabel = QLabel("Username")

self.enterButton = QPushButton("login")

self.signUpButton = QPushButton("Sign Up")

self.loginLayout = QVBoxLayout()

self.loginLayout.addWidget(self.usernameLabel)

self.loginLayout.addWidget(self.usernameBox)

self.loginLayout.addWidget(self.pwordLabel)

self.loginLayout.addWidget(self.pwordBox)

self.loginLayout.addWidget(self.enterButton)

self.loginLayout.addWidget(self.signUpButton)

self.loginFrame.setLayout(self.loginLayout)

self.enterButton.clicked.connect(self.processPword)

self.signUpButton.clicked.connect(self.signUp)

self.thread.printTime.connect(self.printer)

self.thread.drawOps.connect(self.drawOpponents)

self.thread.ended.connect(self.endDialogue)

self.thread.inputTake.connect(self.takeInput)

self.raiseConfirm.clicked.connect(self.returnRaiseValue)

self.raiseTxt.editingFinished.connect(self.updateRaiseSlider)

self.raiseSlider.sliderReleased.connect(self.updateRaiseTxt)

self.quitButton.clicked.connect(self.exitGame)

self.gamelayout = QVBoxLayout()

self.windowLayout = QVBoxLayout()

self.menuLayout = QVBoxLayout()

self.menuLayout.addWidget(self.startButton)

self.menuLayout.addWidget(self.optionsButton)

self.menuLayout.addWidget(self.menuBrowserButton)

self.menuLayout.addWidget(self.quitButton)

self.gamelayout.addWidget(self.printerLabel)

self.gamelayout.addWidget(self.opponentBox)

self.gamelayout.addStretch(1)

self.gamelayout.addLayout(centerRow)

self.gamelayout.addStretch(1)

self.gamelayout.addLayout(handrow)

self.gamelayout.addLayout(self.raiseRow)

self.gamelayout.addWidget(self.selectionCRdo)

self.gamelayout.addWidget(self.selectionRRdo)

self.gamelayout.addWidget(self.selectionFRdo)

self.gamelayout.addWidget(self.buttonConfirm)

self.menuFrame = QFrame()

self.menuFrame.setLayout(self.menuLayout)

self.windowLayout.addWidget(self.menuFrame)

self.menuFrame.hide()

self.gameFrame = QFrame()

self.gameFrame.setLayout(self.gamelayout)

self.windowLayout.addWidget(self.gameFrame)

self.windowLayout.addWidget(self.browserFrame)

self.gameFrame.hide()

# layout.addLayout(self.inputLayout)

self.selectionCRdo.hide()

self.selectionRRdo.hide()

self.selectionFRdo.hide()

self.buttonConfirm.hide()

self.setLayout(self.windowLayout)

self.setWindowTitle(self.tr("Poker Game"))

self.loginFrame.show()

## the password management system

### the idea

Client-side aspects:

* Passwords + usernames plain text are taken in via user interface
* Passwords plain text is asymmetrically encrypted using public key. This is done using the rsa protocol through the cryptography library for python
* Both uname and password are sent to the server with the tag

Server-side aspects:

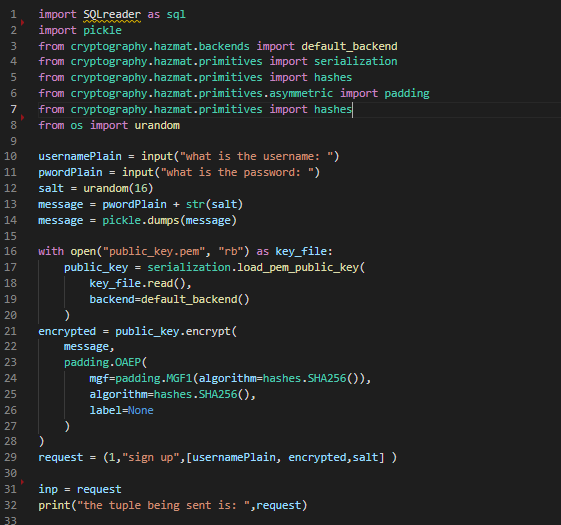
* The username is used to retrieve the hash of the salt and password as well as the salt in plain text
* The password is decrypted using private key and then the salt is applied
* This salted value is then hashed and compared against the hash from the database if it is the same then it returns true

If signing up for the first time the salt is generated client side and the encrypted pword + salt as well as the username and salt in plain text are sent to the db. This is then decrypted hashed and written to the data and then confirmation is returned. Upon receiving this confirmation the client then calls the log in function with the same data they entered to sign up which ensures the sign up process worked.

### The prototype

The prototype was a single file that would simulate the process on one computer by removing the user interface and networking aspects which allows me to rapidly identify and fix issues that come up as well as making the process as efficient and streamlined as possible.

#### The first section (client side sign up)



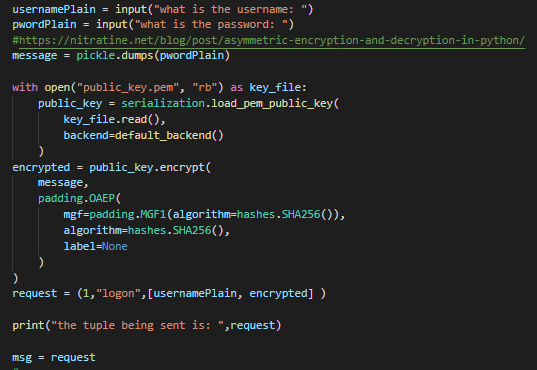
This first block of code has imports for the whole file but crucially it also includes the generation of the salt. This caused a lot of issues as I tried different libraries such as random and bcrypto and cryptography however these all came with their own formats or strange imports which added complications which the os module did not have and the reason I didn’t start with the os module is simply not being aware of the urandom feature. This block starts by taking inputs, then combines them if necessary and ends by pickling them, this step is important as pickling makes things into bytes objects which can then get unpickled into strings even if the original was also a bytes object which causes issues when hashing as the hashing algorithm requires specific data types.

#### Block 2, server side sign up



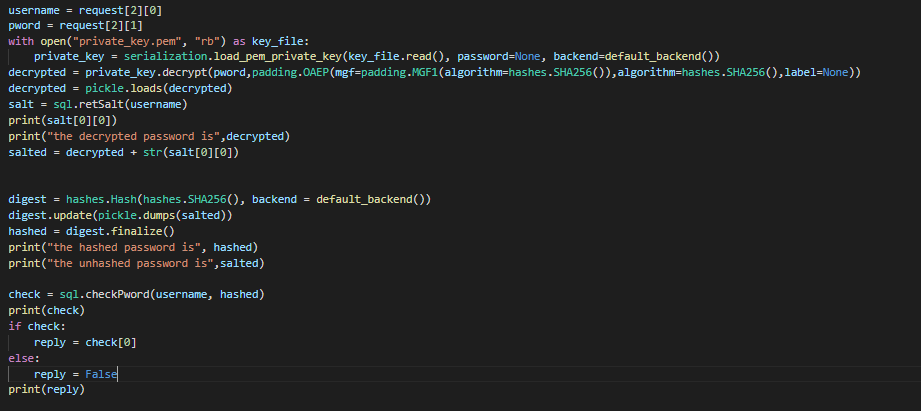
This is a short simple section which decodes and then decrypts the incoming password hashes it using SHA256 hashing algorithm included in the cryptography module for python. This is printed for me to debug and then written to the database.

#### Block 3, client-side login



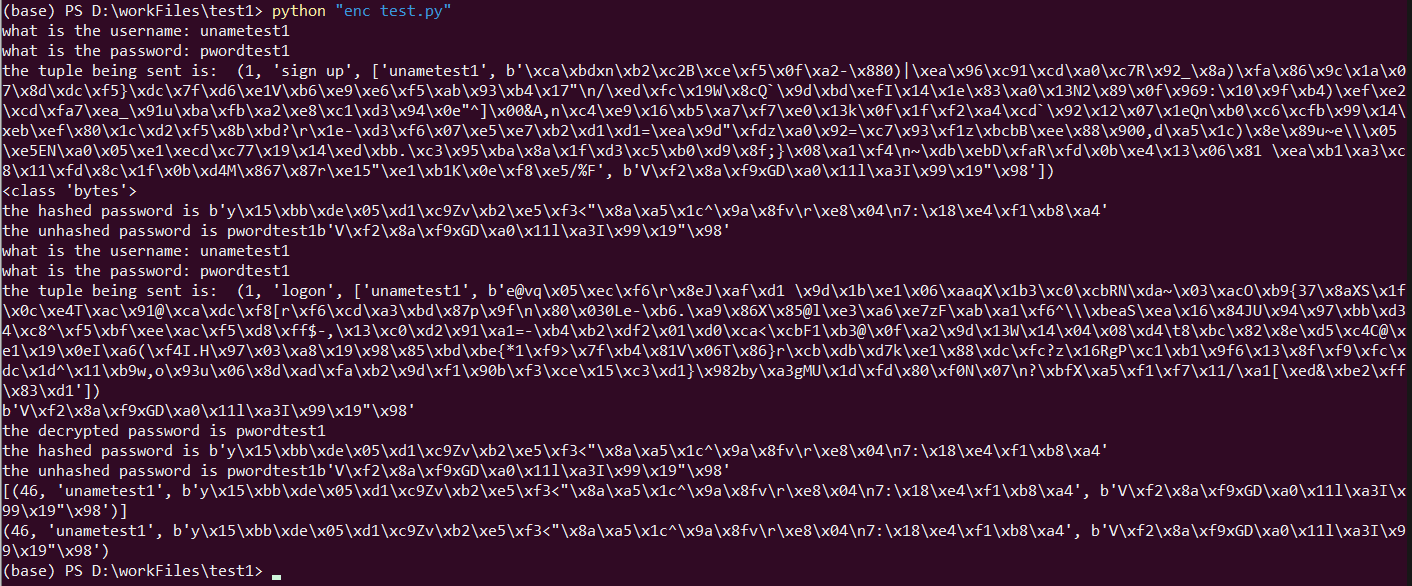
This block takes a second set of inputs, encrypts using the public key and creates a tuple to simulate what would be sent over the network in the actual program. This tuple is also printed for debugging so I can easily understand any issues. This also led to questions such as, how do penguins get to school? The answer of course is on their icycles. Furthermore since this is a prototype and learning experience it is very useful to understand the exact format and what the functions that I used from the cryptography module actually returned.

#### Block 4 server-side login



This last block handles login requests taking the tuple from the last block, decrypts it using the private key, searches the database for the username, returns the salt and adds the salt to the passwords then hashes it and checks it against one stored in the database, if they match then it returns the username of the person, if not it returns False.

#### The output



#### The issues and what I learned.

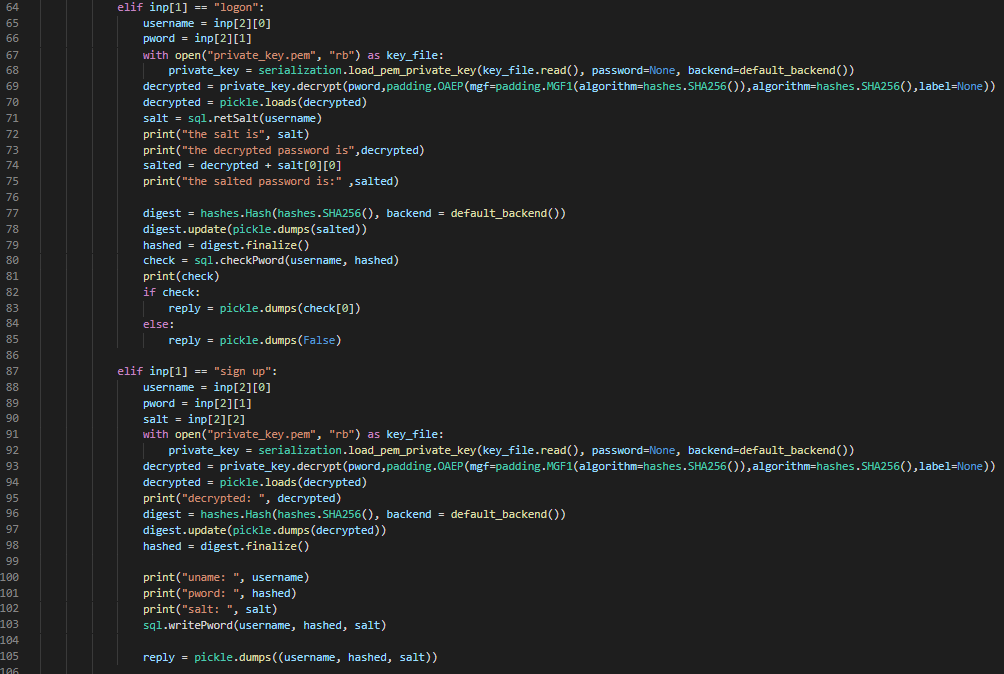
By creating this prototype I familiarised myself with the functions of the cryptography module and allowed me to test the sequence for the program and when I should apply the salt as this was not immediately clear. I also contacted industry professionals about this sequence and they agreed that the salt should be applied server side on checks and client side on sign ups. The main issues I ran into surrounded the salt how to generate it, how to concatenate bytes(the salt) and strings (the password). These were mostly resolved by switching to urandom and properly casting the data types.

#### Necessary changes for the online version

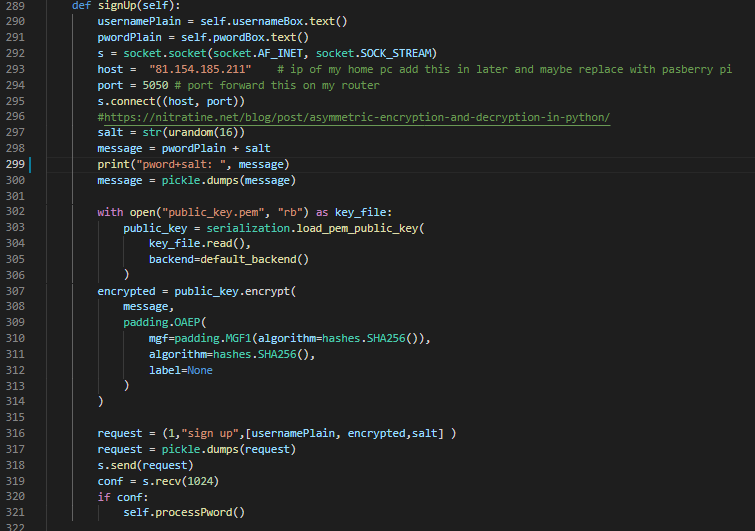
Very few changes were necessary when shifting into the networked version 🐧 and the changes that were necessary were inconsequential and minor changes primarily brought about by the pickling not properly converting to the correct file type leading to even stricter casting.

### The final code

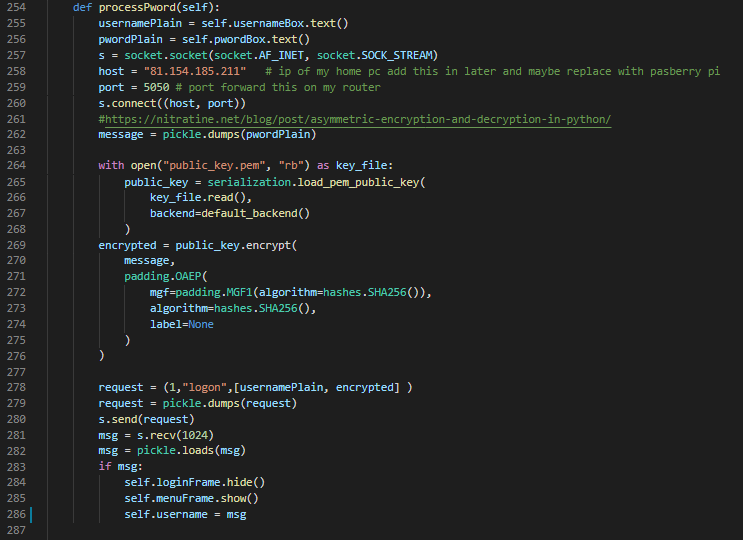
#### Server side



#### Client side sign up.



#### Client-side log in



The most crucial part of this is line 286 which makes a property of the window the username which is what actually determines the name and avoids the issue I had in testing of 2 accounts with the same username causing what were effectively hash collisions with 2 people having the default name.

Note: all ips will be replaced with permanent hosted ips once a permanent server space is bought.