Candidate Number:
3097

Software Development:

6.1: The database

```
# coding=utf-8
import sqlite3

db = sqlite3.connect('clinic.db')
cursor = db.cursor()
"""
creates the entire database with sutiable columns
"""
cursor.execute('''CREATE TABLE IF NOT EXISTS clients(
                ClientID INTEGER PRIMARY KEY AUTOINCREMENT,
                MedicalRecordID INTEGER unique,
                Prefix TEXT,
                FirstName TEXT,
                Surname TEXT,
                DOB TEXT,
                Telephone TEXT,
                Address TEXT,
                Postcode TEXT,
                FOREIGN KEY(MedicalRecordID) REFERENCES medicalrecords(MedicalRecordID))
                ''')

cursor.execute('''CREATE TABLE IF NOT EXISTS staff(
                StaffID INTEGER PRIMARY KEY AUTOINCREMENT,
                Prefix TEXT,
                FirstName TEXT,
                Surname TEXT,
                DOB TEXT,
                Telephone TEXT,
                Address TEXT,
                Postcode TEXT,
                Username TEXT unique,
                Position TEXT,
                LoggedIn BOOLEAN,
                Password TEXT)
                ''')
```

Candidate Number:
3097

```
cursor.execute('''CREATE TABLE IF NOT EXISTS medicalrecords(
                MedicalRecordID INTEGER PRIMARY KEY AUTOINCREMENT,
                ClientID INTEGER unique,
                Sex TEXT,
                Gender TEXT,
                BloodType TEXT,
                Height REAL,
                Mass REAL,
                FOREIGN KEY(ClientID) REFERENCES clients(ClientID))
                ''')

cursor.execute('''CREATE TABLE IF NOT EXISTS appointments(
                AppointmentID INTEGER PRIMARY KEY AUTOINCREMENT,
                ClientID INTEGER,
                StaffID INTEGER,
                TransactionID INTEGER unique,
                StartDateAndTime TEXT,
                EndDateAndTime TEXT,
                AppointmentStatus TEXT,
                FOREIGN KEY(ClientID) REFERENCES clients(ClientID),
                FOREIGN KEY(StaffID) REFERENCES staff(StaffID),
                FOREIGN KEY(TransactionID) REFERENCES transactions(TransactionID))
                ''')

cursor.execute('''CREATE TABLE IF NOT EXISTS transactions(
                TransactionID INTEGER PRIMARY KEY AUTOINCREMENT,
                Difference FLOAT,
                DateAndTime TEXT,
                TransactionStatus TEXT)
                ''')

db.commit()

cursor.execute("""UPDATE staff SET LoggedIn = ?""", ("False",)) #puts loggedin to false

db.commit()
```

Which creates:

Table: appointments

	AppointmentID	ClientID	StaffID	TransactionID	tartDateAndTim	EndDateAndTime	ppointmentStatu
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	3	1	2019-04-04 1...	2019-04-04 1...	Cancelled
2	2	8	4	2	2019-04-04 1...	2019-04-04 1...	Active
3	3	3	8	3	2019-04-04 1...	2019-04-04 1...	Active
4	4	1	9	4	2019-04-05 1...	2019-04-05 1...	Active
5	17	6	8	20	2019-03-11 1...	2019-03-11 1...	Active
6	18	7	3	21	2019-03-11 1...	2019-03-11 1...	Active
7	19	7	8	22	2019-03-12 1...	2019-03-12 1...	Active
8	20	4	4	23	2019-03-09 1...	2019-03-09 1...	Active
9	21	7	4	24	2019-03-13 1...	2019-03-13 1...	Active
10	22	1	8	25	2019-03-13 1...	2019-03-13 1...	Active

Candidate Number:
3097

	ClientID	MedicalRecordID	Prefix	FirstName	Surname	DOB	Telephone	Address	Postcode
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	Mr	Kai	Holloway	1934-07-11	07863701077	23 Ermin Street	RH18 9WX
2	2	2	Mr	Andrew	Rove	1938-04-12	07920991649	19 Middlewich...	TN5 9JE
3	3	3	Mrs	Aaliyah	Price	1978-04-30	07935747494	34 Bishopthor...	SA44 7LU
4	4	4	Ms	Paisley	Russel	2002-08-26	07830280919	12 Sutton Wic...	CA14 1DQ
5	5	5	Mr	Barack	Obama	1997-10-12	07702989653	98 Newport R...	IV12 5WH
6	6	6	Mr	Cathal	Gorman	1954-09-01	07038789458	49 Ash Lane	EX23 7WL
7	7	7	Dr	Abigail	Dunn	1977-08-31	07777827240	5 Manor Close	LL54 0UD
8	8	8	Mr	Leo	Barker	1999-03-16	07956646717	69 Cefn Road	RG7 4FJ
9	9	9	Mrs	Jasmine	Powell	1941-01-04	07728974291	72 South Cres...	DY9 9XX
10	10	10	Mr	Dayyan	OBrien	2001-01-31	07767862981	10 Windsor Hill	BT34 1ER
11	12	12	Dr	Steve	Jobs	1980-01-02	07605407812	5 Apple Lane	RZ7 2GS

	MedicalRecordID	ClientID	Sex	Gender	BloodType	Height	Mass
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	M	Male	B-	1.75	75.0
2	2	2	M	Male	A+	1.73	67.0
3	3	3	F	Female	O-	1.67	60.0
4	4	4	F	Female	O+	1.6	52.0
5	5	5	M	Male	AB+	1.83	101.0
6	6	6	M	Male	B+	1.8	70.0
7	7	7	F	Female	B-	1.65	58.0
8	8	8	M	Male	O-	1.78	64.0
9	9	9	F	Female	AB-	1.54	50.0
10	10	10	M	Male	O+	1.73	58.0
11	12	12	NULL	NULL	NULL	NULL	NULL

	StaffID	Prefix	FirstName	Surname	DOB	Telephone	Address	Postcode	Username	Position	LoggedIn	Password
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Dr	Kirsten	Freener	2001-01-01	07901611233	10 Windsor Hill	BT341ER	owneruser	Owner	1	ownerpass
2	2	Mr	Scott	Randall	1946-04-22	07916795451	92 Sea Road	CO4 3AX	itusernamе	IT	False	itpassword
3	3	Dr	Freya	Winter	1951-08-31	07750976324	1 Foregate St...	KT11 1EF	nurseuser	Nurse	False	nursepass
4	4	Dr	Adam	Pochinki	1966-08-30	07744443297	97 Crown Str...	SO32 5EL	physiouser	Physiotherapist	False	physiopass
5	5	Mr	Jay	Baldwin	1977-01-03	07814692250	77 East Street	SN9 2TX	receptionistuser	Receptionist	False	receptionistpass
6	6	Mrs	Laura	Lyons	1959-10-29	07825964889	14 Warren St	YO8 2NG	llyons886	IT	False	nie7ohteiNu
7	7	Ms	Chloe	Coleman	1973-11-28	07005303213	21 Folestone ...	BA14 0PA	Starman	Nurse	False	chaiwu0H
8	8	Dr	Hayden	Oliver	1972-02-22	07819248729	19 City Walls Rd	SY7 2DD	Agescits1972	Physiotherapist	False	Su5jaht9ie
9	9	Mr	Jamie	Webb	1986-04-25	07842073019	43 Bouverie R...	SY4 3UB	Haters1986	Receptionist	False	phahz1Tee3

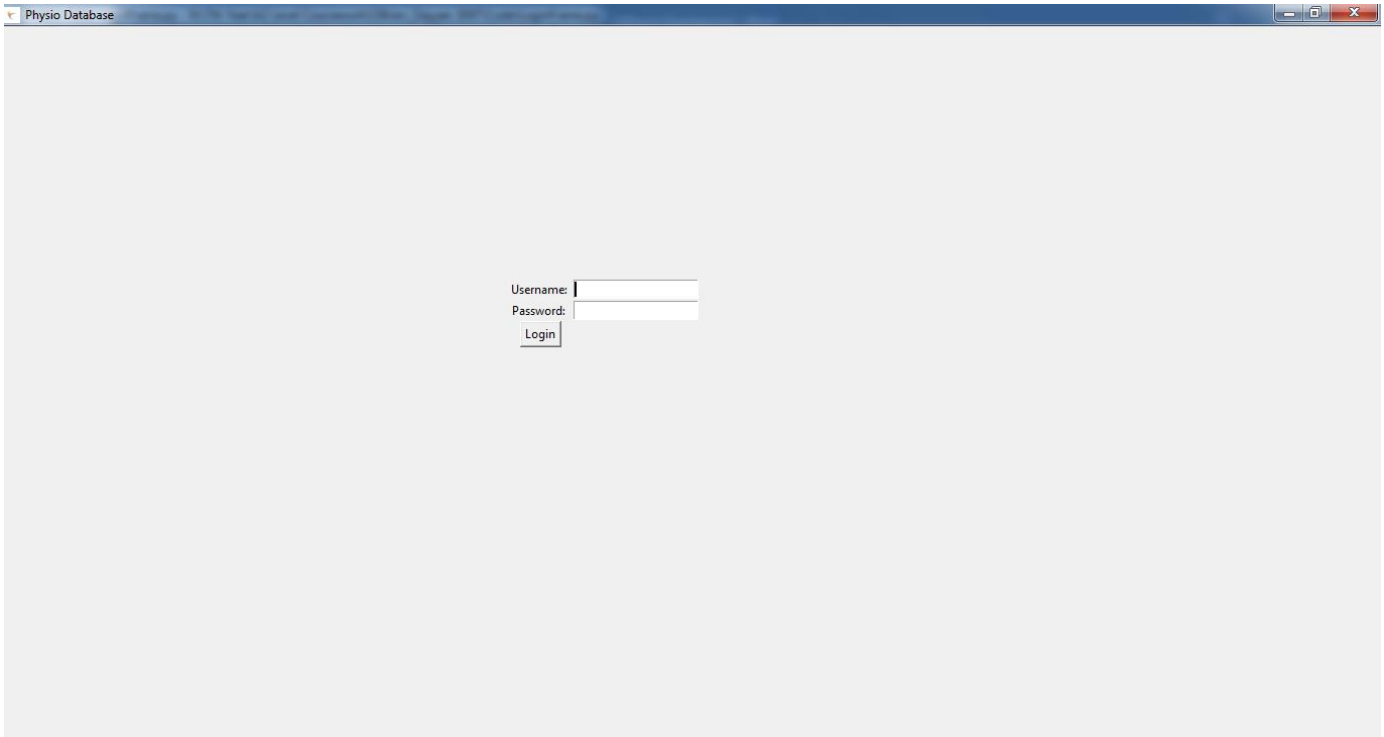
	TransactionID	Difference	DateAndTime	TransactionStatu
	Filter	Filter	Filter	Filter
1	1	60.0	2019-03-04 1...	Sucessful
2	2	25.0	2019-03-04 1...	Sucessful
3	3	25.0	2019-03-04 1...	Sucessful
4	4	50.0	2019-03-04 1...	Sucessful
5	19	56.0	2019-03-11 1...	Successful
6	20	50.0	2019-03-11 1...	Successful
7	21	57.0	2019-03-11 1...	Successful
8	22	75.0	2019-03-11 1...	Successful
9	23	60.0	2019-03-11 1...	Successful
10	24	55.0	2019-03-11 1...	Successful
11	25	75.0	2019-03-11 1...	Successful
12	26	-500.0	2019-03-11 1...	Successful

Candidate Number:
3097

6.2: Login screen GUI

```
class LoginFrame(tk.Frame):
    """
    creates the login page frame, allows the user to input the username
    and password and when the button is pressed it switches the frame
    to HomepageTab and shows all the tab options
    """

    def __init__(self, master):
        tk.Frame.__init__(self, master)
        self.master = master
        tk.Label(self, text="Username: ").grid(row=0, column=0, padx = (500,0), pady = (250,0))
        tk.Label(self, text="Password: ").grid(row=1, column=0, padx = (500,0))
        self.un_entry = tk.Entry(self)
        self.un_entry.grid(row=0, column=1, padx = (0,500), pady = (250,0))
        self.pw_entry = tk.Entry(self, show = '*') #use stars for password
        self.pw_entry.grid(row=1, column=1, padx = (0,500))

        self.username = self.un_entry.get()
        self.password = self.pw_entry.get()

        tk.Button(self, text="Login", command=self.check_login).grid(row=2, column=0, padx = (500,0), pady = (0,250))
```

Which creates:



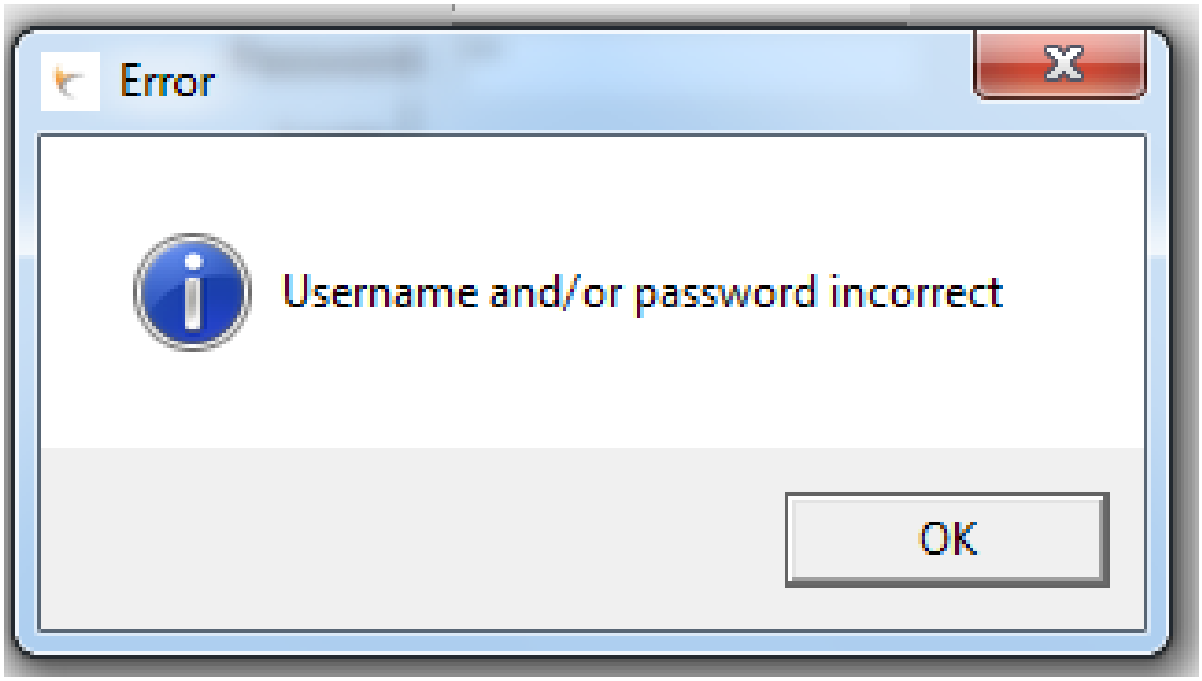
6.3: Login screen frame switcher

```
# coding=utf-8

from AppointmentsTab import *
from ClientsTab import *
from FinancesTab import *
from MedicalRecordsTab import *
from StaffTab import *
from database_setup import *


class SwitchFrame(tk.Tk):
    """
    the initially run class, it creates the ability
    frame of the window without creating a new one
    as well as making the initial frame LoginPage
    """

    def __init__(self):
        tk.Tk.__init__(self)
        self._frame = None
        self.switch_frame(LoginFrame) #puts login fr
        self.geometry("1275x750")
        self.title("Physio Database")
        self.iconbitmap('logo.ico')

    def switch_frame(self, frame_class):
        """destroys the current frame and replaces i
        new_frame = frame_class(self)
        if self._frame is not None:
            self._frame.destroy() #destroys the logi
        self._frame = new_frame
        self._frame.grid(row=0, column=0)
```

6.4: Login screen login confirmation

```
def check_login(self):
    """
    checks if the username and password is
    correct and then sets LoggedIn = True
    """

    username = self.un_entry.get()
    password = self.password = self.pw_entry.get()
    cursor.execute("""SELECT LoggedIn FROM staff WHERE Username = ? AND Password = ?""", (username, password))
    result = cursor.fetchone()
    if result:
        cursor.execute("""UPDATE staff SET LoggedIn = ? WHERE Username = ? AND Password = ?""",
                       (True, username, password))
        db.commit()
        self.master.switch_frame(TabFrame) #puts tab frame on top
    else:
        messagebox.showinfo("Error", "Username and/or password incorrect")
```

Which creates:



If the password is incorrect, otherwise it will bring them to the client's page

Candidate Number:
3097

6.5: Login screen views

```
class TabFrame(tk.Frame):
    """
    this is the frame which holds all the tabs, this allows users
    to select which tab they want to use
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)

        notebook = ttk.Notebook(parent)

        cursor.execute("""SELECT Position FROM staff WHERE LoggedIn = ?""",(True,))
        pos = cursor.fetchone()[0] #does views to see which tabs to show
        if pos == 'IT                    '  or pos == 'Owner                 ' or pos == 'Nurse                ' or pos == 'Physiotherapist' or pos == 'Receptionist
            notebook.add(ClientsFrame(notebook), text='Clients') #shows client tab
        if pos == 'Owner                 ' or pos == 'Nurse                ' or pos == 'Physiotherapist' or pos == 'Receptionist     ':
            notebook.add(AppointmentsFrame(notebook), text='Appointments')
        if pos == 'IT                    '  or pos == 'Owner                 ':
            notebook.add(StaffFrame(notebook), text='Staff')
        if pos == 'Owner                 ' or pos == 'Nurse                ' or pos == 'Physiotherapist':
            notebook.add(MedicalRecordsFrame(notebook), text='Medical Records')
        if pos == 'Owner                 ':
            notebook.add(FinancesFrame(notebook), text='Finances')

        notebook.grid(row=0, column=0)


if __name__ == "__main__":
    SwitchFrame().mainloop()
```

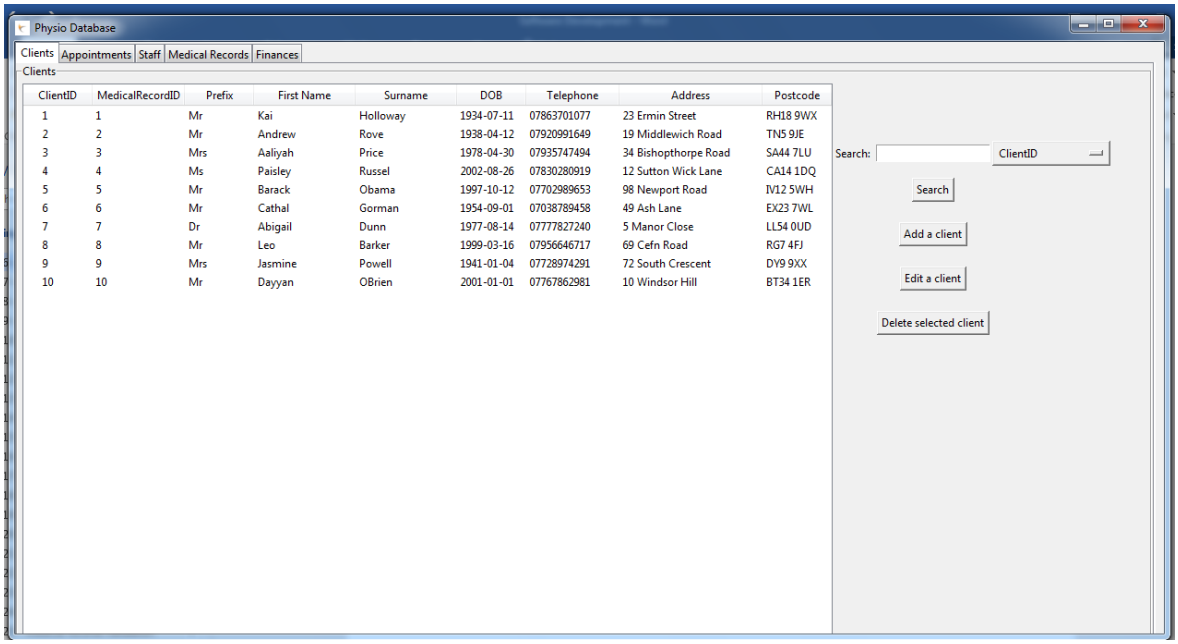
6.6: Client tab GUI

```
# coding=utf-8

import sqlite3
import tkinter as Tkinter
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import *
from tkinter import messagebox
import ast


class ClientsFrame(tk.Frame):
    """
    the frame for the clients tab
    """

    def __init__(self, parent, *args, **kwargs):
        """
        intially run function, sets up design of the entire page
        """
        tk.Frame.__init__(self, parent, *args, **kwargs)

        clients_frame = tk.LabelFrame(self, text="Clients", padx=5, pady=5, width=1275, height=750)
        clients_frame.grid(row=0, column=0)

        OPTIONS = [
            "ClientID                ",
            "MedicalRecordID",
            "Prefix                    ",
            "First Name            ",
            "Surname                ",
            "DOB                        ",
            "Telephone            ",
            "Address                ",
            "Postcode                ",
        ] #creates the options for the dropdown menu on serach

        variable = StringVar(clients_frame)
        variable.set(OPTIONS[0]) #starts the dropdown menu on ClientID
```

Candidate Number:
3097

```
clients_frame.tree = ttk.Treeview(clients_frame, height="33", selectmode='browse',
                                  columns=(
                                      'MedicalRecordID', 'Prefix', 'First Name', 'Surname', 'DOB', 'Telephone', 'Address',
                                      'Postcode')) #creates the treeview
ClientsFrame.tree = clients_frame.tree
clients_frame.tree.heading('#0', text='ClientID')
clients_frame.tree.heading('#1', text='MedicalRecordID')
clients_frame.tree.heading('#2', text='Prefix')
clients_frame.tree.heading('#3', text='First Name')
clients_frame.tree.heading('#4', text='Surname')
clients_frame.tree.heading('#5', text='DOB')
clients_frame.tree.heading('#6', text='Telephone')
clients_frame.tree.heading('#7', text='Address')
clients_frame.tree.heading('#8', text='Postcode')
clients_frame.tree.column('#0', stretch=Tkinter.YES, width="75", minwidth="50")
clients_frame.tree.column('#1', stretch=Tkinter.YES, width="100", minwidth="100")
clients_frame.tree.column('#2', stretch=Tkinter.YES, width="75", minwidth="50")
clients_frame.tree.column('#3', stretch=Tkinter.YES, width="110", minwidth="85")
clients_frame.tree.column('#4', stretch=Tkinter.YES, width="110", minwidth="85")
clients_frame.tree.column('#5', stretch=Tkinter.YES, width="75", minwidth="75")
clients_frame.tree.column('#6', stretch=Tkinter.YES, width="100", minwidth="100")
clients_frame.tree.column('#7', stretch=Tkinter.YES, width="155", minwidth='100')
clients_frame.tree.column('#8', stretch=Tkinter.YES, width="75", minwidth="75")
clients_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
clients_frame.treeview = clients_frame.tree

search = tk.Label(clients_frame, text="Search: ")
search.grid(row=10, column=50)
search_box = clients_frame.search_entry = tk.Entry(clients_frame)
search_box.grid(row=10, column=51)
dropdownsearch = OptionMenu(clients_frame, variable, *OPTIONS)
dropdownsearch.grid(row=10, column=52)

tk.Button(clients_frame, text="Search", command=self.search_table).grid(row=11, column=51)

self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()


 self.cursor.execute("""SELECT Position FROM staff WHERE LoggedIn = ?""",(True,))
 pos = self.cursor.fetchone()[0]
 if pos == 'Owner                ' or pos == 'Nurse                ' or pos == 'Physiotherapist' or pos == 'Receptionist     ': #
     tk.Button(clients_frame, text="Add a client", command=CreateClientFrame).grid(row=13, column=51)

     tk.Button(clients_frame, text="Edit a client", command=EditClientFrame).grid(row=15, column=51)

     tk.Button(clients_frame, text="Delete selected client", command=self.delete_client).grid(row=17, column=51)

 pad = tk.Label(clients_frame, text="")
 pad.grid(row=10, column=55, padx=(0, 30))

 self.tree = clients_frame.tree
 self.variable = variable
 self.search_box = search_box
 self.update_table()
```

Which creates:



Candidate Number:
3097

6.7: Update clients table

```
def update_table(self):
    """
    updates the treeview and fills it with all records
    in the ClientID table
    """
    self.cursor.execute("""SELECT * FROM clients""")
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children()) #clears table
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:])
```

6.8: Search clients

```
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "ClientID                ":
        self.cursor.execute("""SELECT * FROM clients WHERE ClientID LIKE ?""", ('%' + search + '%',))
        drop_down = "ClientID"
    elif drop_down == "MedicalRecordID":
        self.cursor.execute("""SELECT * FROM clients WHERE MedicalRecordID LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Prefix                    ":
        self.cursor.execute("""SELECT * FROM clients WHERE Prefix LIKE ?""", ('%' + search + '%',))
        drop_down = "Prefix"
    elif drop_down == "First Name            ":
        self.cursor.execute("""SELECT * FROM clients WHERE FirstName LIKE ?""", ('%' + search + '%',))
        drop_down = "First Name"
    elif drop_down == "Surname               ":
        self.cursor.execute("""SELECT * FROM clients WHERE Surname LIKE ?""", ('%' + search + '%',))
        drop_down = "Surname"
    elif drop_down == "DOB                       ":
        self.cursor.execute("""SELECT * FROM clients WHERE DOB LIKE ?""", ('%' + search + '%',))
        drop_down = "DOB"
    elif drop_down == "Telephone            ":
        self.cursor.execute("""SELECT * FROM clients WHERE Telephone LIKE ?""", ('%' + search + '%',))
        drop_down = "Telephone"
    elif drop_down == "Address                 ":
        self.cursor.execute("""SELECT * FROM clients WHERE Address LIKE ?""", ('%' + search + '%',))
        drop_down = "Address"
    elif drop_down == "Postcode                ":
        self.cursor.execute("""SELECT * FROM clients WHERE Postcode LIKE ?""", ('%' + search + '%',))
        drop_down = "Postcode"
    self.tree.delete(*self.tree.get_children()) #clears treeview table
    rows = self.cursor.fetchall()
    f = open('clients.txt','w')
    f.write("------------------------------------- Searched for '" + search + "' by " + drop_down + "-------------------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "ClientID: " + str(row[0]))
        f.write('\n' + "MedicalRecordID: "  + str(row[1]))
        f.write('\n' + "Prefix: " + str(row[2]))
        f.write('\n' + "First Name: " + str(row[3]))
        f.write('\n' + "Surname: " + str(row[4]))
        f.write('\n' + "DOB: " + str(row[5]))
        f.write('\n' + "Telephone: " + str(row[6]))
        f.write('\n' + "Address: " + str(row[7]))
        f.write('\n' + "Postcode: " + str(row[8]))
        f.write('\n')
    messagebox.showinfo("Alert", "Clients saved (clients.txt)")
    f.close()
```

Candidate Number:
3097

Which creates:

```
|------------------------------------- Searched for '' by ClientID-------------------------------------

ClientID: 1
MedicalRecordID: 1
Prefix: Mr
First Name: Kai
Surname: Holloway
DOB: 1934-07-11
Telephone: 07863701077
Address: 23 Ermin Street
Postcode: RH18 9WX

ClientID: 2
MedicalRecordID: 2
Prefix: Mr
First Name: Andrew
Surname: Rove
DOB: 1938-04-12
Telephone: 07920991649
Address: 19 Middlewich Road
Postcode: TN5 9JE

ClientID: 3
MedicalRecordID: 3
Prefix: Mrs
First Name: Aaliyah
Surname: Price
DOB: 1978-04-30
Telephone: 07935747494
Address: 34 Bishopthorpe Road
Postcode: SA44 7LU

ClientID: 4
MedicalRecordID: 4
Prefix: Ms
First Name: Paisley
Surname: Russel
DOB: 2002-08-26
Telephone: 07830280919
Address: 12 Sutton Wick Lane
Postcode: CA14 1DQ

ClientID: 5
MedicalRecordID: 5
Prefix: Mr
First Name: Barack
Surname: Obama
DOB: 1997-10-12
Telephone: 07702989653
Address: 98 Newport Road
Postcode: IV12 5WH
```

Candidate Number:
3097

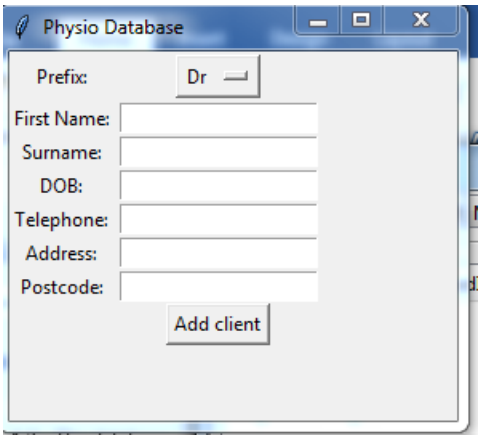
6.9: Create client's GUI

```
class CreateClientFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new client,
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = ClientsFrame.tree
        create_client_window = tk.Toplevel(self)
        create_client_window.geometry("280x230") #creates the new window with geomtery so yo

        PREFIXOPTIONS = [
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(create_client_window)
        variable.set(PREFIXOPTIONS[0])

        prefix = tk.Label(create_client_window, text="Prefix: ")
        prefix.grid(row=0, column=0)
        dropdownsearch = OptionMenu(create_client_window, variable, *PREFIXOPTIONS)
        dropdownsearch.grid(row=0, column=1)

        first_name = tk.Label(create_client_window, text="First Name: ")
        first_name.grid(row=1, column=0)
        first_name_box = create_client_window.search_entry = tk.Entry(create_client_window)
        first_name_box.grid(row=1, column=1)

        surname = tk.Label(create_client_window, text="Surname: ")
        surname.grid(row=2, column=0)
        surname_box = create_client_window.search_entry = tk.Entry(create_client_window)
        surname_box.grid(row=2, column=1)

        dob = tk.Label(create_client_window, text="DOB: ")
        dob.grid(row=3, column=0)
        dob_box = create_client_window.search_entry = tk.Entry(create_client_window)
        dob_box.grid(row=3, column=1)

        telephone = tk.Label(create_client_window, text="Telephone: ")
        telephone.grid(row=4, column=0)
```

Candidate Number:
3097

```
address = tk.Label(create_client_window, text="Address: ")
address.grid(row=5, column=0)
address_box = create_client_window.search_entry = tk.Entry(create_client_window)
address_box.grid(row=5, column=1)

postcode = tk.Label(create_client_window, text="Postcode: ")
postcode.grid(row=6, column=0)
postcode_box = create_client_window.search_entry = tk.Entry(create_client_window)
postcode_box.grid(row=6, column=1)

search_button = tk.Button(create_client_window, text="Add client", command=self.add_client)
search_button.grid(row=7, column=1)

self.variable = variable
self.first_name_box = first_name_box
self.surname_box = surname_box
self.dob_box = dob_box
self.telephone_box = telephone_box
self.address_box = address_box
self.postcode_box = postcode_box
self.create_client_window = create_client_window
self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()
```

Which creates:



6.10: Edit client's GUI

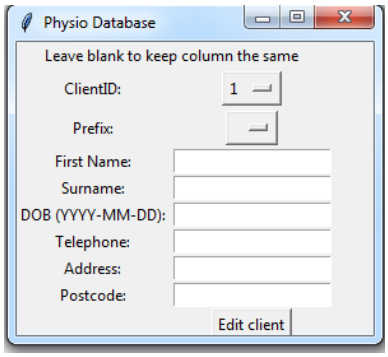
```
class EditClientFrame(tk.Frame):
    """
    creates a new window and allows the user to edit a current client,
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = ClientsFrame.tree
        edit_client_window = tk.Toplevel(self)
        edit_client_window.geometry("280x230")
        self.edit_client_window = edit_client_window

        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        PREFIXOPTIONS = [
            "",
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(edit_client_window)
        variable.set(PREFIXOPTIONS[0])

        self.cursor.execute("""SELECT ClientID FROM clients""")

        CLIENTOPTIONS = []
        result = self.cursor.fetchall()
        for item in result:
            CLIENTOPTIONS.append(item)

        clientvariable = StringVar(edit_client_window)

        try:
            clientvariable.set(CLIENTOPTIONS[0]) #checks if there are any clients alrea
        except IndexError:
            messagebox.showinfo("Error", "No clients available")
            self.edit_client_window.destroy()

        psa = tk.Label(edit_client_window, text="Leave blank to keep column the same ")
        psa.grid(row=0, column=0, columnspan=2)
```

Candidate Number:
3097

```
clientid = tk.Label(edit_client_window, text="ClientID: ")
clientid.grid(row=1, column=0)
clientidsearch = OptionMenu(edit_client_window, clientvariable, *CLIENTOPTIONS)
clientidsearch.grid(row=1, column=1)

prefix = tk.Label(edit_client_window, text="Prefix: ")
prefix.grid(row=2, column=0)
dropdownsearch = OptionMenu(edit_client_window, variable, *PREFIXOPTIONS)
dropdownsearch.grid(row=2, column=1)

first_name = tk.Label(edit_client_window, text="First Name: ")
first_name.grid(row=3, column=0)
first_name_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
first_name_box.grid(row=3, column=1)

surname = tk.Label(edit_client_window, text="Surname: ")
surname.grid(row=4, column=0)
surname_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
surname_box.grid(row=4, column=1)

dob = tk.Label(edit_client_window, text="DOB (YYYY-MM-DD): ")
dob.grid(row=5, column=0)
dob_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
dob_box.grid(row=5, column=1)

telephone = tk.Label(edit_client_window, text="Telephone: ")
telephone.grid(row=6, column=0)
telephone_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
telephone_box.grid(row=6, column=1)

address = tk.Label(edit_client_window, text="Address: ")
address.grid(row=7, column=0)
address_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
address_box.grid(row=7, column=1)

postcode = tk.Label(edit_client_window, text="Postcode: ")
postcode.grid(row=8, column=0)
postcode_box = edit_client_window.search_entry = tk.Entry(edit_client_window)
postcode_box.grid(row=8, column=1)

search_button = tk.Button(edit_client_window, text="Edit client", command=self.add_client)
search_button.grid(row=9, column=1)
```

```
self.variable = variable
self.clientvariable = clientvariable
self.first_name_box = first_name_box
self.surname_box = surname_box
self.dob_box = dob_box
self.telephone_box = telephone_box
self.address_box = address_box
self.postcode_box = postcode_box

self.postcode_box.bind("<Return>", self.add_client)
```

Which creates:



Candidate Number:
3097

6.11: Delete clients

```
def delete_client(self):
    """
    deletes highlighted client
    """
    iid_selected = self.tree.focus() #finds the iid of the highlighted box
    client_id = self.tree.item(iid_selected, 'text') #gets the client id from the iid of the

    self.cursor.execute("""DELETE from clients WHERE ClientID = ? """, (client_id,))
    self.cursor.execute("""DELETE from medicalrecords WHERE ClientID = ? """, (client_id,))
    self.cursor.execute("""DELETE from appointments WHERE ClientID = ? """, (client_id,))
    self.db.commit()
    self.update_table()
```

6.12: Client validation

```
def add_client(self):
    """
    checks if add client results are valid and then updates tables
    """
    if self.first_name_box.get().isalpha():
        if 20 > len(self.first_name_box.get()) > 2:
            if self.surname_box.get().isalpha():
                if 20 > len(self.surname_box.get()) > 2:
                    pattern = r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'
                    match = re.search(pattern, self.dob_box.get())
                    if match:
                        if 10 < len(self.telephone_box.get()) < 12:
                            if self.telephone_box.get().isdigit():
                                if 50 > len(self.address_box.get()) > 5:
                                    pattern = r'(GIR|([A-Z-[QVX][0-9][0-9]?)|(([A-Z-[QVX][A-Z-[IJZ][0-9][0-9]?)|(([A-Z-[QVX][0-9][A-HJKSTUW])|([A-Z-[QVX][A-Z-[IJZ][0-9][ABE
                                    match = re.search(pattern, self.postcode_box.get())
                                    if match:
                                        self.cursor.execute(
                                            """INSERT INTO clients(Prefix, FirstName, Surname, DOB, Telephone, Address, Postcode) VALUES (?,?,?,?,?,?,?)""",
                                            (self.variable.get(), self.first_name_box.get(), self.surname_box.get(),
                                             self.dob_box.get(), self.telephone_box.get(), self.address_box.get(),
                                             self.postcode_box.get(),))
                                        self.cursor.execute(
                                            """SELECT ClientID FROM clients ORDER BY ClientID DESC LIMIT 1"""
                                        )
                                        clientid = self.cursor.fetchone()
                                        clientid = clientid[0]
                                        self.cursor.execute(
                                            """INSERT INTO medicalrecords(ClientID) VALUES (?)""", (clientid,))
                                        self.cursor.execute(
                                            """SELECT MedicalRecordID FROM medicalrecords ORDER BY MedicalRecordID DESC LIMIT 1"""
                                        )
                                        medicalrecordid = self.cursor.fetchone()
                                        medicalrecordid = medicalrecordid[0]
                                        self.cursor.execute(
                                            """UPDATE clients SET MedicalRecordID = ? WHERE ClientID = ?""",
                                            (medicalrecordid, clientid,))
                                        self.db.commit()
                                        self.create_client_window.destroy()
                                        ClientsFrame.update_table(self)
                                    else:
                                        messagebox.showinfo("Error", "Postcode incorrect format")
                                else:
                                    messagebox.showinfo("Error", "Address is an invalid length")
                            else:
                                messagebox.showinfo("Error", "Telephone number contains non numeric characters")
                        else:
                            messagebox.showinfo("Error", "Telephone number is an invalid length. Must be 11 digits")
                    else:
                        messagebox.showinfo("Error", "DOB incorrect format (YYYY-MM-DD)")
                else:
                    messagebox.showinfo("Error", "Surname is an invalid length")
            else:
                messagebox.showinfo("Error", "Surname contains non alphabetical characters")
        else:
            messagebox.showinfo("Error", "First name is an invalid length")
    else:
        messagebox.showinfo("Error", "First name contains non alphabetical characters")
```

Candidate Number:
3097

```
def add_client(self):
    """
    checks if add client results are valid and then updates tables
    """

    clientvariable = self.clientvariable.get()
    clientvariable = (ast.literal_eval(clientvariable)[0])  # converts to tuple

    if self.variable.get() == "":
        pass
    else:
        self.cursor.execute("""UPDATE clients SET Prefix = ? WHERE ClientID = ?""",
                            (self.variable.get(), clientvariable,))

    if self.first_name_box.get() == "":
        pass
    elif self.first_name_box.get().isalpha():
        if 20 > len(self.first_name_box.get()) > 2:
            self.cursor.execute("""UPDATE clients SET FirstName = ? WHERE ClientID = ?""",
                                (self.first_name_box.get(), clientvariable,))
        else:
            messagebox.showinfo("Error", "First name is an invalid length")
    else:
        messagebox.showinfo("Error", "First name contains non alphabetical characters")

    if self.surname_box.get() == "":
        pass
    elif self.surname_box.get().isalpha():
        if 20 > len(self.surname_box.get()) > 2:
            self.cursor.execute("""UPDATE clients SET Surname = ? WHERE ClientID = ?""",
                                (self.surname_box.get(), clientvariable,))
        else:
            messagebox.showinfo("Error", "Surname is an invalid length")
    else:
        messagebox.showinfo("Error", "Surname contains non alphabetical characters")

    if self.dob_box.get() == "":
        pass
    else:
        pattern = r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'
        match = re.search(pattern, self.dob_box.get())
        if match:
            self.cursor.execute("""UPDATE clients SET DOB = ? WHERE ClientID = ?""",
                                (self.dob_box.get(), clientvariable,))
        else:
            messagebox.showinfo("Error", "DOB incorrect format")
```

```
if self.telephone_box.get() == "":
    pass
elif 10 < len(self.telephone_box.get()) < 12:
    if self.telephone_box.get().isdigit():
        self.cursor.execute("""UPDATE clients SET Telephone = ? WHERE ClientID = ?""",
                            (self.telephone_box.get(), clientvariable,))
    else:
        messagebox.showinfo("Error", "Telephone number contains non numeric characters")
else:
    messagebox.showinfo("Error", "Telephone number is an invalid length. Must be 11 digits")

if self.address_box.get() == "":
    pass
elif 50 > len(self.address_box.get()) > 5:
    self.cursor.execute("""UPDATE clients SET Address = ? WHERE ClientID = ?""",
                        (self.address_box.get(), clientvariable,))
else:
    messagebox.showinfo("Error", "Address is an invalid length")

if self.postcode_box.get() == "":
    pass
else:
    pattern = r'(GIR|([A-Z-[QVX][0-9][0-9]?)|(([A-Z-[QVX][A-Z-[IJZ][0-9][0-9]?)|(([A-Z-[QVX][0-9][A-HJKSTUW])|([A-Z-[QVX][A-Z-[IJZ][0-9][ABEHMNPRVWXY])))(?=( )?[0-9][A-
    match = re.search(pattern, self.postcode_box.get())
    if match:
        self.cursor.execute("""UPDATE clients SET Postcode = ? WHERE ClientID = ?""",
                            (self.postcode_box.get(), clientvariable,))
    else:
        messagebox.showinfo("Error", "Postcode incorrect format")

self.db.commit()
self.edit_client_window.destroy()
ClientsFrame.update_table(self)
```

Candidate Number:
3097

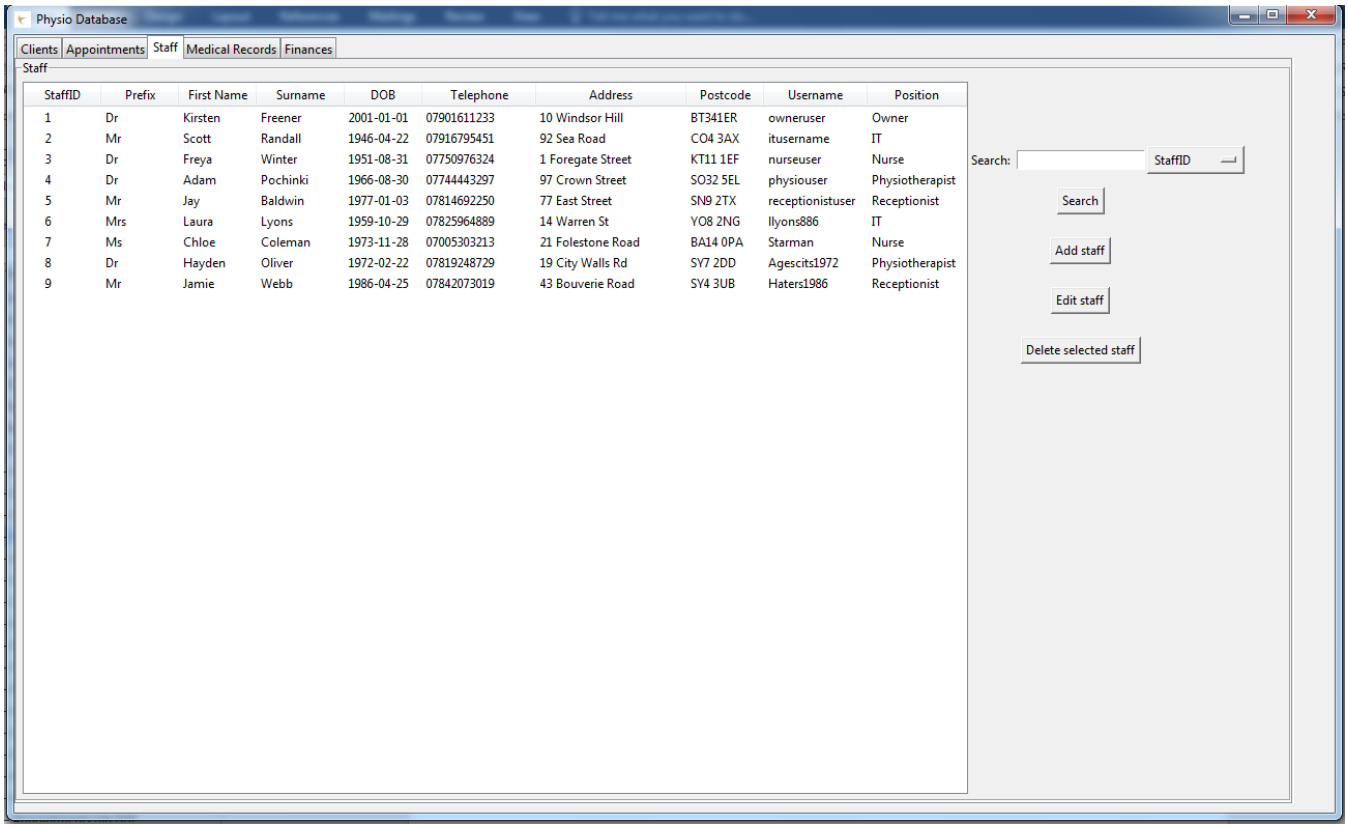
6.13: Staff tab GUI

```
# coding=utf-8

import tkinter as Tkinter
import tkinter.ttk as ttk
import re
import sqlite3
from tkinter import *
import tkinter as tk
import ast
from tkinter import messagebox


class StaffFrame(tk.Frame):
    """
    the frame for the finances tab
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)
        staff_frame = tk.LabelFrame(self, text="Staff", padx=5, pady=5, width=1275, height=750)
        staff_frame.grid(row=0, column=0)

        OPTIONS = [
            "StaffID      ",
            "Prefix           ",
            "First Name",
            "Surname    ",
            "DOB             ",
            "Telephone",
            "Address      ",
            "Postcode    ",
            "Username  ",
            "Position   "
        ]

        variable = StringVar(staff_frame)
        variable.set(OPTIONS[0])

        # Set the treeview
        staff_frame.tree = ttk.Treeview(staff_frame, height="33", selectmode='browse',
                                        columns=(
                                            'Prefix', 'First Name', 'Surname', 'DOB', 'Telephone', 'Address',
                                            'Postcode', 'Username', 'Position'))
        StaffFrame.tree = staff_frame.tree
        staff_frame.tree.heading('#0', text='StaffID')
        staff_frame.tree.heading('#1', text='Prefix')
        staff_frame.tree.heading('#2', text='First Name')
        staff_frame.tree.heading('#3', text='Surname')
        staff_frame.tree.heading('#4', text='DOB')
```

Candidate Number:
3097

```
        staff_frame.tree.heading('#5', text='Telephone')
        staff_frame.tree.heading('#6', text='Address')
        staff_frame.tree.heading('#7', text='Postcode')
        staff_frame.tree.heading('#8', text='Username')
        staff_frame.tree.heading('#9', text='Position')
        staff_frame.tree.column('#0', stretch=Tkinter.YES, width="75", minwidth="50")
        staff_frame.tree.column('#1', stretch=Tkinter.YES, width="75", minwidth="50")
        staff_frame.tree.column('#2', stretch=Tkinter.YES, width="75", minwidth="50")
        staff_frame.tree.column('#3', stretch=Tkinter.YES, width="85", minwidth="50")
        staff_frame.tree.column('#4', stretch=Tkinter.YES, width="75", minwidth="75")
        staff_frame.tree.column('#5', stretch=Tkinter.YES, width="110", minwidth="100")
        staff_frame.tree.column('#6', stretch=Tkinter.YES, width="145", minwidth='100')
        staff_frame.tree.column('#7', stretch=Tkinter.YES, width="75", minwidth="75")
        staff_frame.tree.column('#8', stretch=Tkinter.YES, width="100", minwidth='100')
        staff_frame.tree.column('#9', stretch=Tkinter.YES, width="95", minwidth="50")
        staff_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
        staff_frame.treeview = staff_frame.tree

        search = tk.Label(staff_frame, text="Search: ")
        search.grid(row=10, column=50)
        search_box = staff_frame.search_entry = tk.Entry(staff_frame)
        search_box.grid(row=10, column=51)
        dropdownsearch = OptionMenu(staff_frame, variable, *OPTIONS)
        dropdownsearch.grid(row=10, column=52)

        tk.Button(staff_frame, text="Search", command=self.search_table).grid(row=11, column=51)

        tk.Button(staff_frame, text="Add staff", command=CreateStaffFrame).grid(row=13, column=51)

        tk.Button(staff_frame, text="Edit staff", command=EditStaffFrame).grid(row=15, column=51)

        tk.Button(staff_frame, text="Delete selected staff", command=self.delete_staff).grid(row=17, column=51)

        pad = tk.Label(staff_frame, text="")
        pad.grid(row=10, column=55, padx=(0, 30))

        self.tree = staff_frame.tree
        self.variable = variable
        self.search_box = search_box
        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()
        self.update_table()
```

Which creates:



StaffID	Prefix	First Name	Surname	DOB	Telephone	Address	Postcode	Username	Position
1	Dr	Kirsten	Freener	2001-01-01	07901611233	10 Windsor Hill	BT341ER	owneruser	Owner
2	Mr	Scott	Randall	1946-04-22	07916795451	92 Sea Road	CO4 3AX	itusername	IT
3	Dr	Freya	Winter	1951-08-31	07750976324	1 Foregate Street	KT11 1EF	nurseuser	Nurse
4	Dr	Adam	Pochinki	1966-08-30	07744443297	97 Crown Street	SO32 5EL	physiouser	Physiotherapist
5	Mr	Jay	Baldwin	1977-01-03	07814692250	77 East Street	SN9 2TX	receptionistuser	Receptionist
6	Mrs	Laura	Lyons	1959-10-29	07825964889	14 Warren St	YO8 2NG	llyons886	IT
7	Ms	Chloe	Coleman	1973-11-28	07005303213	21 Folestone Road	BA14 0PA	Starman	Nurse
8	Dr	Hayden	Oliver	1972-02-22	07819248729	19 City Walls Rd	SY7 2DD	Agescits1972	Physiotherapist
9	Mr	Jamie	Webb	1986-04-25	07842073019	43 Bouverie Road	SY4 3UB	Haters1986	Receptionist

Candidate Number:
3097

6.14: Update staff table

```
def update_table(self):
    """
    updates the treeview and fills it with all records
    in the staffid table
    """
    self.cursor.execute("""SELECT * FROM staff""")
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:10])
```

6.15: Search staff

```
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "StaffID      ":
        self.cursor.execute("""SELECT * FROM staff WHERE StaffID LIKE ?""", ('%' + search + '%',))
        drop_down = "StaffID"
    elif drop_down == "Prefix          ":
        self.cursor.execute("""SELECT * FROM staff WHERE Prefix LIKE ?""", ('%' + search + '%',))
        drop_down = "Prefix"
    elif drop_down == "First Name":
        self.cursor.execute("""SELECT * FROM staff WHERE FirstName LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Surname    ":
        self.cursor.execute("""SELECT * FROM staff WHERE Surname LIKE ?""", ('%' + search + '%',))
        drop_down = "Surname"
    elif drop_down == "DOB            ":
        self.cursor.execute("""SELECT * FROM staff WHERE DOB LIKE ?""", ('%' + search + '%',))
        drop_down = "DOB"
    elif drop_down == "Telephone":
        self.cursor.execute("""SELECT * FROM staff WHERE Telephone LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Address      ":
        self.cursor.execute("""SELECT * FROM staff WHERE Address LIKE ?""", ('%' + search + '%',))
        drop_down = "Address"
    elif drop_down == "Postcode   ":
        self.cursor.execute("""SELECT * FROM staff WHERE Postcode LIKE ?""", ('%' + search + '%',))
        drop_down = "Postcode"
    elif drop_down == "Username  ":
        self.cursor.execute("""SELECT * FROM staff WHERE Username LIKE ?""", ('%' + search + '%',))
        drop_down = "Username"
    elif drop_down == "Position   ":
        self.cursor.execute("""SELECT * FROM staff WHERE Position LIKE ?""", ('%' + search + '%',))
        drop_down = "Position"
    self.tree.delete(*self.tree.get_children())
    rows = self.cursor.fetchall()
    f = open('staff.txt','w')
    f.write("------------------------------------- Searched for '" + search + "' by " + drop_down + "-------------------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "StaffID: " + str(row[0]))
        f.write('\n' + "Prefix: " + str(row[1]))
        f.write('\n' + "First Name: " + str(row[2]))
        f.write('\n' + "Surname: " + str(row[3]))
        f.write('\n' + "DOB: " + str(row[4]))
        f.write('\n' + "Telephone: " + str(row[5]))
        f.write('\n' + "Address: " + str(row[6]))
        f.write('\n' + "Postcode: " + str(row[7]))
        f.write('\n' + "Username: " + str(row[8]))
        f.write('\n' + "Position: " + str(row[9]))
        f.write('\n')
    messagebox.showinfo("Alert", "Staff saved (staff.txt)")
    f.close()
```

Candidate Number:
3097

Which creates:

```
|------------------------------------- Searched for '' by StaffID-------------------------------------

StaffID: 1
Prefix: Dr
First Name: Kirsten
Surname: Freener
DOB: 2001-01-01
Telephone: 07901611233
Address: 10 Windsor Hill
Postcode: BT341ER
Username: owneruser
Position: Owner

StaffID: 2
Prefix: Mr
First Name: Scott
Surname: Randall
DOB: 1946-04-22
Telephone: 07916795451
Address: 92 Sea Road
Postcode: CO4 3AX
Username: itusername
Position: IT

StaffID: 3
Prefix: Dr
First Name: Freya
Surname: Winter
DOB: 1951-08-31
Telephone: 07750976324
Address: 1 Foregate Street
Postcode: KT11 1EF
Username: nurseuser
Position: Nurse

StaffID: 4
Prefix: Dr
First Name: Adam
Surname: Pochinki
DOB: 1966-08-30
Telephone: 07744443297
Address: 97 Crown Street
Postcode: SO32 5EL
Username: physiouser
Position: Physiotherapist

StaffID: 5
Prefix: Mr
First Name: Jay
Surname: Baldwin
DOB: 1977-01-03
Telephone: 07814692250
Address: 77 East Street
Postcode: SN9 2TX
```

Candidate Number:
3097

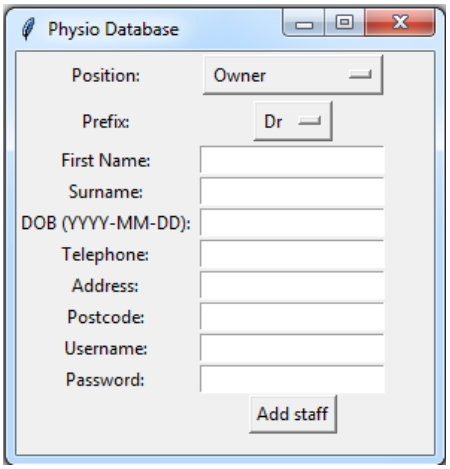
6.16: Create staff GUI

```
class CreateStaffFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new staff,
    when completed it will update the treeview in staffTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = StaffFrame.tree
        create_staff_window = tk.Toplevel(self)
        create_staff_window.geometry("280x270")

        PREFIXOPTIONS = [
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(create_staff_window)
        variable.set(PREFIXOPTIONS[0])

        POSITIONOPTIONS = [
            "Owner                ",
            "IT                     ",
            "Nurse                ",
            "Physiotherapist",
            "Receptionist     "
        ]

        positionvariable = StringVar(create_staff_window)
        positionvariable.set(POSITIONOPTIONS[0])

        position = tk.Label(create_staff_window, text="Position: ")
        position.grid(row=0, column=0)
        positiondropdownsearch = OptionMenu(create_staff_window, positionvariable, *POSITIONOPTIONS)
        positiondropdownsearch.grid(row=0, column=1)

        prefix = tk.Label(create_staff_window, text="Prefix: ")
        prefix.grid(row=1, column=0)
        dropdownsearch = OptionMenu(create_staff_window, variable, *PREFIXOPTIONS)
        dropdownsearch.grid(row=1, column=1)
```

Candidate Number:
3097

```
first_name = tk.Label(create_staff_window, text="First Name: ")
first_name.grid(row=2, column=0)
first_name_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
first_name_box.grid(row=2, column=1)

surname = tk.Label(create_staff_window, text="Surname: ")
surname.grid(row=3, column=0)
surname_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
surname_box.grid(row=3, column=1)

dob = tk.Label(create_staff_window, text="DOB (YYYY-MM-DD): ")
dob.grid(row=4, column=0)
dob_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
dob_box.grid(row=4, column=1)

telephone = tk.Label(create_staff_window, text="Telephone: ")
telephone.grid(row=5, column=0)
telephone_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
telephone_box.grid(row=5, column=1)

address = tk.Label(create_staff_window, text="Address: ")
address.grid(row=6, column=0)
address_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
address_box.grid(row=6, column=1)

postcode = tk.Label(create_staff_window, text="Postcode: ")
postcode.grid(row=7, column=0)
postcode_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
postcode_box.grid(row=7, column=1)

username = tk.Label(create_staff_window, text="Username: ")
username.grid(row=8, column=0)
username_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
username_box.grid(row=8, column=1)

password = tk.Label(create_staff_window, text="Password: ")
password.grid(row=9, column=0)
password_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
password_box.grid(row=9, column=1)

search_button = tk.Button(create_staff_window, text="Add staff", command=self.add_staff)
search_button.grid(row=10, column=1)

self.variable = variable
self.first_name_box = first_name_box
self.surname_box = surname_box
self.dob_box = dob_box
self.telephone_box = telephone_box
self.address_box = address_box
self.postcode_box = postcode_box
self.positionvariable = positionvariable
self.username_box = username_box
self.password_box = password_box
self.create_staff_window = create_staff_window
self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()
```

Which creates:



6.17: Edit staff GUI

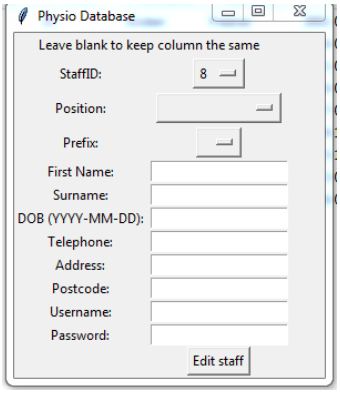
```
class EditStaffFrame(tk.Frame):
    """
    creates a new window and allows the user to edit a current s
    when completed it will update the treeview in staffTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = StaffFrame.tree
        edit_staff_window = tk.Toplevel(self)
        edit_staff_window.geometry("280x310")
        self.edit_staff_window = edit_staff_window

        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        PREFIXOPTIONS = [
            "",
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(edit_staff_window)
        variable.set(PREFIXOPTIONS[0])

        self.cursor.execute("""SELECT StaffID FROM staff""")

        STAFFOPTIONS = []
        result = self.cursor.fetchall()
        for item in result:
            STAFFOPTIONS.append(item)

        staffvariable = StringVar(edit_staff_window)

        try:
            staffvariable.set(STAFFOPTIONS[0])
        except IndexError:
            messagebox.showinfo("Error", "No staff available")
            self.edit_staff_window.destroy()
```

Candidate Number:
3097

```
POSITIONOPTIONS = [
    "                        ",
    "Owner                ",
    "IT                     ",
    "Nurse                ",
    "Physiotherapist",
    "Receptionist    "
]

positionvariable = StringVar(edit_staff_window)
positionvariable.set(POSITIONOPTIONS[0])

psa = tk.Label(edit_staff_window, text="Leave blank to keep column the same ")
psa.grid(row=0, column=0, columnspan=2)

staffid = tk.Label(edit_staff_window, text="StaffID: ")
staffid.grid(row=1, column=0)
staffidsearch = OptionMenu(edit_staff_window, staffvariable, *STAFFOPTIONS)
staffidsearch.grid(row=1, column=1)

position = tk.Label(edit_staff_window, text="Position: ")
position.grid(row=2, column=0)
positiondropdownsearch = OptionMenu(edit_staff_window, positionvariable, *POSITIONOPTIONS)
positiondropdownsearch.grid(row=2, column=1)

prefix = tk.Label(edit_staff_window, text="Prefix: ")
prefix.grid(row=3, column=0)
dropdownsearch = OptionMenu(edit_staff_window, variable, *PREFIXOPTIONS)
dropdownsearch.grid(row=3, column=1)

first_name = tk.Label(edit_staff_window, text="First Name: ")
first_name.grid(row=4, column=0)
first_name_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
first_name_box.grid(row=4, column=1)

surname = tk.Label(edit_staff_window, text="Surname: ")
surname.grid(row=5, column=0)
surname_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
surname_box.grid(row=5, column=1)

dob = tk.Label(edit_staff_window, text="DOB (YYYY-MM-DD): ")
dob.grid(row=6, column=0)
dob_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
dob_box.grid(row=6, column=1)

telephone = tk.Label(edit_staff_window, text="Telephone: ")
telephone.grid(row=7, column=0)
telephone_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
telephone_box.grid(row=7, column=1)
```

```
address = tk.Label(edit_staff_window, text="Address: ")
address.grid(row=8, column=0)
address_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
address_box.grid(row=8, column=1)

postcode = tk.Label(edit_staff_window, text="Postcode: ")
postcode.grid(row=9, column=0)
postcode_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
postcode_box.grid(row=9, column=1)

username = tk.Label(edit_staff_window, text="Username: ")
username.grid(row=10, column=0)
username_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
username_box.grid(row=10, column=1)

password = tk.Label(edit_staff_window, text="Password: ")
password.grid(row=11, column=0)
password_box = edit_staff_window.search_entry = tk.Entry(edit_staff_window)
password_box.grid(row=11, column=1)

search_button = tk.Button(edit_staff_window, text="Edit staff", command=self.add_staff)
search_button.grid(row=12, column=1)

self.variable = variable
self.staffvariable = staffvariable
self.first_name_box = first_name_box
self.surname_box = surname_box
self.dob_box = dob_box
self.telephone_box = telephone_box
self.address_box = address_box
self.postcode_box = postcode_box
self.positionvariable = positionvariable
self.username_box = username_box
self.password_box = password_box

self.postcode_box.bind("<Return>", self.add_staff)
```

Which creates:



6.18: Delete staff

```
def delete_staff(self):
    """
    deletes highlighted staff
    """
    iid_selected = self.tree.focus()
    staff_id = self.tree.item(iid_selected, 'text')

    self.cursor.execute("""DELETE from staff WHERE StaffID = ? """, (staff_id,))
    self.db.commit()
    self.update_table()
```

Candidate Number:
3097

6.19: Staff validation

```
def add_staff(self):
    """
    checks if add staff results are valid and then updates tables
    """
    if self.first_name_box.get().isalpha():
        if 20 > len(self.first_name_box.get()) > 2:
            if self.surname_box.get().isalpha():
                if 20 > len(self.surname_box.get()) > 2:
                    pattern = r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'
                    match = re.search(pattern, self.dob_box.get())
                    if match:
                        if 10 < len(self.telephone_box.get()) < 12:
                            if self.telephone_box.get().isdigit():
                                if 50 > len(self.address_box.get()) > 5:
                                    pattern = r'(GIR|([A-Z-[QVX][0-9][0-9]?)|(([A-Z-[QVX][A-Z-[IJZ][0-9][0-9]?)|(([A-Z-[QVX][0-9][A-HJKSTUW])|([A-Z-[QVX][A-Z-[IJZ][0-9][ABE
                                    match = re.search(pattern, self.postcode_box.get())
                                    if match:
                                        if len(self.username_box.get()) > 6:
                                            if len(self.password_box.get()) > 6:
                                                try:
                                                    self.cursor.execute(
                                                        """INSERT INTO staff(Prefix, FirstName, Surname, DOB, Telephone, Address,
                                                            Postcode, Username, Position, LoggedIn, Password) VALUES (?,?,?,?,?,?,?,?,?,?,?)""",
                                                        (self.variable.get(), self.first_name_box.get(),
                                                         self.surname_box.get(),
                                                         self.dob_box.get(), self.telephone_box.get(),
                                                         self.address_box.get(),
                                                         self.postcode_box.get(), self.username_box.get(),
                                                         self.positionvariable.get(), False,
                                                         self.password_box.get()))
                                                    self.db.commit()
                                                    self.create_staff_window.destroy()
                                                    StaffFrame.update_table(self)
                                                except sqlite3.IntegrityError:
                                                    messagebox.showinfo("Error", "Username not unique")
                                            else:
                                                messagebox.showinfo("Error", "Password too short")
                                        else:
                                            messagebox.showinfo("Error", "Username too short")
                                    else:
                                        messagebox.showinfo("Error", "Postcode incorrect format")
                                else:
                                    messagebox.showinfo("Error", "Address is an invalid length")


                        else:
                            messagebox.showinfo("Error", "Telephone number contains non numeric characters")
                    else:
                        messagebox.showinfo("Error", "Telephone number is an invalid length. Must be 11 digits")
                else:
                    messagebox.showinfo("Error", "DOB incorrect format")
            else:
                messagebox.showinfo("Error", "Surname is an invalid length")
        else:
            messagebox.showinfo("Error", "Surname contains non alphabetical characters")
    else:
        messagebox.showinfo("Error", "First name is an invalid length")
else:
    messagebox.showinfo("Error", "First name contains non alphabetical characters")
```

Candidate Number:
3097

```python
def add_staff(self):
    """
    checks if add staff results are valid and then updates tables
    """
    staffvariable = self.staffvariable.get()
    staffvariable = (ast.literal_eval(staffvariable)[0])  # converts to tuple

    if self.variable.get() == "":
        pass
    else:
        self.cursor.execute("""UPDATE staff SET Prefix = ? WHERE StaffID = ?""",
                            (self.variable.get(), staffvariable,))

    if self.positionvariable.get() == "                        ":
        pass
    else:
        self.cursor.execute("""UPDATE staff SET Position = ? WHERE StaffID = ?""",
                            (self.positionvariable.get(), staffvariable,))

    if self.first_name_box.get() == "":
        pass
    elif self.first_name_box.get().isalpha():
        if 20 > len(self.first_name_box.get()) > 2:
            self.cursor.execute("""UPDATE staff SET FirstName = ? WHERE StaffID = ?""",
                                (self.first_name_box.get(), staffvariable,))
        else:
            messagebox.showinfo("Error", "First name is an invalid length")
    else:
        messagebox.showinfo("Error", "First name contains non alphabetical characters")

    if self.surname_box.get() == "":
        pass
    elif self.surname_box.get().isalpha():
        if 20 > len(self.surname_box.get()) > 2:
            self.cursor.execute("""UPDATE staff SET Surname = ? WHERE StaffID = ?""",
                                (self.surname_box.get(), staffvariable,))
        else:
            messagebox.showinfo("Error", "Surname is an invalid length")
    else:
        messagebox.showinfo("Error", "Surname contains non alphabetical characters")
```

```python
if self.dob_box.get() == "":
    pass
else:
    pattern = r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'
    match = re.search(pattern, self.dob_box.get())
    if match:
        self.cursor.execute("""UPDATE staff SET DOB = ? WHERE StaffID = ?""",
                            (self.dob_box.get(), staffvariable,))
    else:
        messagebox.showinfo("Error", "DOB incorrect format")

if self.telephone_box.get() == "":
    pass
elif 10 < len(self.telephone_box.get()) < 12:
    if self.telephone_box.get().isdigit():
        self.cursor.execute("""UPDATE staff SET Telephone = ? WHERE StaffID = ?""",
                            (self.telephone_box.get(), staffvariable,))
    else:
        messagebox.showinfo("Error", "Telephone number contains non numeric characters")
else:
    messagebox.showinfo("Error", "Telephone number is an invalid length. Must be 11 digits")

if self.address_box.get() == "":
    pass
elif 50 > len(self.address_box.get()) > 5:
    self.cursor.execute("""UPDATE staff SET Address = ? WHERE StaffID = ?""",
                        (self.address_box.get(), staffvariable,))
else:
    messagebox.showinfo("Error", "Address is an invalid length")

if self.postcode_box.get() == "":
    pass
else:
    pattern = r'(GIR|([A-Z-[QVX][0-9][0-9]?)|(([A-Z-[QVX][A-Z-[IJZ][0-9][0-9]?)|(([A-Z-[QVX][0-9][A-HJKSTUW])|([A-Z-[QVX][A-Z-[IJZ][0-9][ABEHMNPRVWXY]))))(?=( )?[0-9][A
    match = re.search(pattern, self.postcode_box.get())
    if match:
        self.cursor.execute("""UPDATE staff SET Postcode = ? WHERE StaffID = ?""",
                            (self.postcode_box.get(), staffvariable,))
    else:
        messagebox.showinfo("Error", "Postcode incorrect format")


if self.password_box.get() == "":
    pass
elif len(self.password_box.get()) > 6:
    self.cursor.execute("""UPDATE staff SET Password = ? WHERE StaffID = ?""",
                        (self.password_box.get(), staffvariable,))
else:
    messagebox.showinfo("Error", "Password too short")

self.db.commit()
self.edit_staff_window.destroy()
StaffFrame.update_table(self)
```

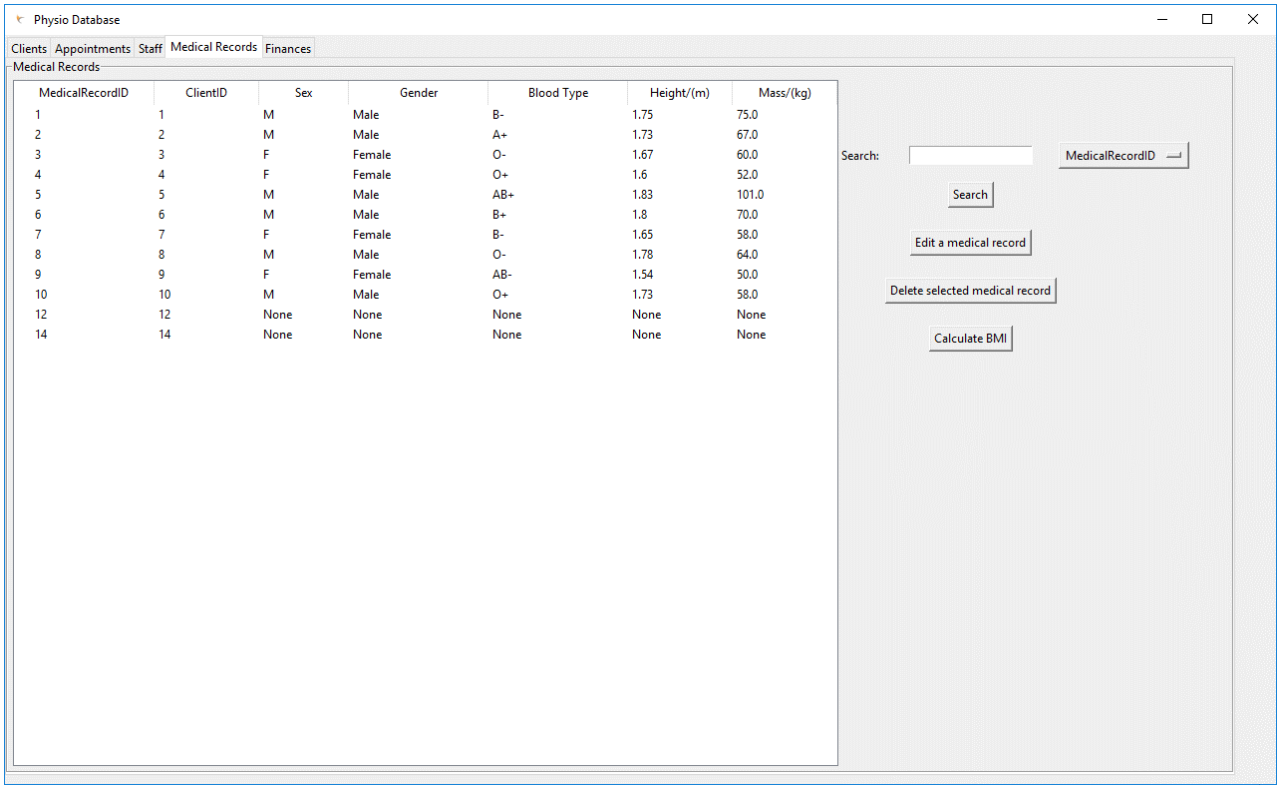
Candidate Number:
3097

6.20: Medical records tab GUI

```
# coding=utf-8

import tkinter as Tkinter
import tkinter.ttk as ttk
import re
import sqlite3
from tkinter import *
import tkinter as tk
from tkinter import messagebox
import ast


class MedicalRecordsFrame(tk.Frame):
    """
    the frame for the medical records tab
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)
        medical_records_frame = tk.LabelFrame(self, text="Medical Records", padx=5, pady=5, width=1000, height=750)
        medical_records_frame.grid(row=0, column=0)

        notebook = ttk.Notebook(parent)

        OPTIONS = [
            "MedicalRecordID",
            "ClientID                    ",
            "Sex                           ",
            "Gender                      ",
            "Blood Type              ",
            "Height                        ",
            "Mass                          "
        ]

        variable = StringVar(medical_records_frame)
        variable.set(OPTIONS[0])

        # Set the treeview
        medical_records_frame.tree = ttk.Treeview(medical_records_frame, height="33", selectmode='browse',
                                                  columns=(
                                                      'ClientID', 'Sex', 'Gender', 'Blood Type', 'Height/(m)',
                                                      'Mass/(kg)'))
        MedicalRecordsFrame.tree = medical_records_frame.tree
        medical_records_frame.tree.heading('#0', text='MedicalRecordID')
        medical_records_frame.tree.heading('#1', text='ClientID')
        medical_records_frame.tree.heading('#2', text='Sex')
        medical_records_frame.tree.heading('#3', text='Gender')
        medical_records_frame.tree.heading('#4', text='Blood Type')
        medical_records_frame.tree.heading('#5', text='Height/(m)')
        medical_records_frame.tree.heading('#6', text='Mass/(kg)')
        medical_records_frame.tree.column('#0', stretch=Tkinter.YES, width="140", minwidth="100")
        medical_records_frame.tree.column('#1', stretch=Tkinter.YES, width="105", minwidth="50")
        medical_records_frame.tree.column('#2', stretch=Tkinter.YES, width="90", minwidth="50")
        medical_records_frame.tree.column('#3', stretch=Tkinter.YES, width="140", minwidth="85")
        medical_records_frame.tree.column('#4', stretch=Tkinter.YES, width="140", minwidth="85")
        medical_records_frame.tree.column('#5', stretch=Tkinter.YES, width="105", minwidth="75")
        medical_records_frame.tree.column('#6', stretch=Tkinter.YES, width="105", minwidth="75")
        medical_records_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
        medical_records_frame.treeview = medical_records_frame.tree

        search = tk.Label(medical_records_frame, text="Search: ")
        search.grid(row=10, column=50)
        search_box = medical_records_frame.search_entry = tk.Entry(medical_records_frame)
        search_box.grid(row=10, column=51)
        dropdownsearch = OptionMenu(medical_records_frame, variable, *OPTIONS)
        dropdownsearch.grid(row=10, column=52)

        tk.Button(medical_records_frame, text="Search", command=self.search_table).grid(row=11, column=51)

        tk.Button(medical_records_frame, text="Edit a medical record", command=EditMedicalRecordFrame).grid(row=13,
                                                                                                            column=51)

        tk.Button(medical_records_frame, text="Delete selected medical record", command=self.delete_client).grid(row=15,
                                                                                                                  column=51)

        tk.Button(medical_records_frame, text="Calculate BMI", command=self.calculate_bmi).grid(row=17, column=51)

        pad = tk.Label(medical_records_frame, text="")
        pad.grid(row=10, column=55, padx=(0, 30))

        self.tree = medical_records_frame.tree
        self.variable = variable
        self.search_box = search_box
        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()
        self.update_table()
```

Candidate Number:
3097

Which creates:



6.21: Update medical records table

```
def update_table(self):
    """
    updates the treeview and fills it with all records
    in the ClientID table
    """
    self.cursor.execute("""SELECT * FROM medicalrecords""")
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:])
```

Candidate Number:
3097

6.22: Search medical records

```python
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "MedicalRecordID":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE MedicalRecordID LIKE ?""", ('%' + search + '%',))
    elif drop_down == "ClientID                ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE ClientID LIKE ?""", ('%' + search + '%',))
        drop_down = "ClientID"
    elif drop_down == "Sex                        ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE Sex LIKE ?""", ('%' + search + '%',))
        drop_down = "Sex"
    elif drop_down == "Gender                  ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE Gender LIKE ?""", ('%' + search + '%',))
        drop_down = "Gender"
    elif drop_down == "Blood Type            ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE BloodType LIKE ?""", ('%' + search + '%',))
        drop_down = "Blood Type"
    elif drop_down == "Height                    ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE Height LIKE ?""", ('%' + search + '%',))
        drop_down = "Height"
    elif drop_down == "Mass                       ":
        self.cursor.execute("""SELECT * FROM medicalrecords WHERE Mass LIKE ?""", ('%' + search + '%',))
        drop_down = "Mass"
    self.tree.delete(*self.tree.get_children())
    rows = self.cursor.fetchall()
    f = open('medicalrecords.txt','w')
    f.write("------------------------------------- Searched for '" + search + "' by " + drop_down + "-------------------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "MedicalRecordID: " + str(row[0]))
        f.write('\n' + "ClientID: "  + str(row[1]))
        f.write('\n' + "Sex: " + str(row[2]))
        f.write('\n' + "Gender: " + str(row[3]))
        f.write('\n' + "Blood Type: " + str(row[4]))
        f.write('\n' + "Height: " + str(row[5]))
        f.write('\n' + "Mass: " + str(row[6]))
        f.write('\n')
    messagebox.showinfo("Alert", "Medical records saved (medicalrecords.txt)")
    f.close()
```

Which creates:

```
------------------------------------- Searched for '' by MedicalRecordID-------------------------------------

MedicalRecordID: 1
ClientID: 1
Sex: M
Gender: Male
Blood Type: B-
Height: 1.75
Mass: 75.0

MedicalRecordID: 2
ClientID: 2
Sex: M
Gender: Male
Blood Type: A+
Height: 1.73
Mass: 67.0

MedicalRecordID: 3
ClientID: 3
Sex: F
Gender: Female
Blood Type: O-
Height: 1.67
Mass: 60.0

MedicalRecordID: 4
ClientID: 4
Sex: F
Gender: Female
Blood Type: O+
Height: 1.6
Mass: 52.0

MedicalRecordID: 5
ClientID: 5
Sex: M
Gender: Male
Blood Type: AB+
Height: 1.83
Mass: 101.0

MedicalRecordID: 6
ClientID: 6
Sex: M
Gender: Male
Blood Type: B+
Height: 1.8
Mass: 70.0
```

Candidate Number:
3097

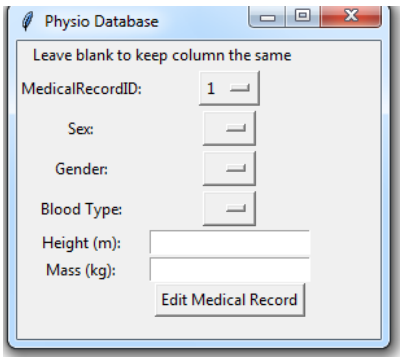
6.23: Edit medical records GUI

```
class EditMedicalRecordFrame(tk.Frame):
    """
    creates a new window and allows the user to edit a current client
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = MedicalRecordsFrame.tree
        edit_medical_record_window = tk.Toplevel(self)
        edit_medical_record_window.geometry("280x230")
        self.edit_medical_record_window = edit_medical_record_window

        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        SEXOPTIONS = [
            "",
            "M",
            "F"
        ]

        sexvariable = StringVar(edit_medical_record_window)
        sexvariable.set(SEXOPTIONS[0])

        GENDEROPTIONS = [
            "",
            "Male",
            "Female",
            "Other"
        ]

        gendervariable = StringVar(edit_medical_record_window)
        gendervariable.set(GENDEROPTIONS[0])

        BLOODTYPEOPTIONS = [
            "",
            "A+",
            "A-",
            "B+",
            "B-",
            "O+",
            "O-",
            "AB+",
            "AB-"
        ]
```

```
bloodtypevariable = StringVar(edit_medical_record_window)
bloodtypevariable.set(BLOODTYPEOPTIONS[0])

self.cursor.execute("""SELECT MedicalRecordID FROM medicalrecords""")

MEDICALRECORDOPTIONS = []
result = self.cursor.fetchall()
for item in result:
    MEDICALRECORDOPTIONS.append(item)

medicalrecordvariable = StringVar(edit_medical_record_window)

try:
    medicalrecordvariable.set(MEDICALRECORDOPTIONS[0])
except IndexError:
    messagebox.showinfo("Error", "No medical records available")
    self.edit_medical_record_window.destroy()

psa = tk.Label(edit_medical_record_window, text="Leave blank to keep column the same ")
psa.grid(row=0, column=0, columnspan=2)

medicalrecordid = tk.Label(edit_medical_record_window, text="MedicalRecordID: ")
medicalrecordid.grid(row=1, column=0)
medicalrecordidsearch = OptionMenu(edit_medical_record_window, medicalrecordvariable, *MEDICALRECORDOPTIONS)
medicalrecordidsearch.grid(row=1, column=1)

sex = tk.Label(edit_medical_record_window, text="Sex: ")
sex.grid(row=2, column=0)
sexsearch = OptionMenu(edit_medical_record_window, sexvariable, *SEXOPTIONS)
sexsearch.grid(row=2, column=1)

gender = tk.Label(edit_medical_record_window, text="Gender: ")
gender.grid(row=3, column=0)
gendersearch = OptionMenu(edit_medical_record_window, gendervariable, *GENDEROPTIONS)
gendersearch.grid(row=3, column=1)

bloodtype = tk.Label(edit_medical_record_window, text="Blood Type: ")
bloodtype.grid(row=4, column=0)
bloodtypesearch = OptionMenu(edit_medical_record_window, bloodtypevariable, *BLOODTYPEOPTIONS)
bloodtypesearch.grid(row=4, column=1)

height = tk.Label(edit_medical_record_window, text="Height (m): ")
height.grid(row=5, column=0)
height_box = edit_medical_record_window.search_entry = tk.Entry(edit_medical_record_window)
height_box.grid(row=5, column=1)
```

```
mass = tk.Label(edit_medical_record_window, text="Mass (kg): ")
mass.grid(row=6, column=0)
mass_box = edit_medical_record_window.search_entry = tk.Entry(edit_medical_record_window)
mass_box.grid(row=6, column=1)

search_button = tk.Button(edit_medical_record_window, text="Edit Medical Record",
                          command=self.add_medical_record)
search_button.grid(row=7, column=1)

self.medicalrecordvariable = medicalrecordvariable
self.sexvariable = sexvariable
self.gendervariable = gendervariable
self.bloodtypevariable = bloodtypevariable
self.height_box = height_box
self.mass_box = mass_box

self.mass_box.bind("<Return>", self.add_medical_record)
```

Which creates:



6.24: Delete medical records

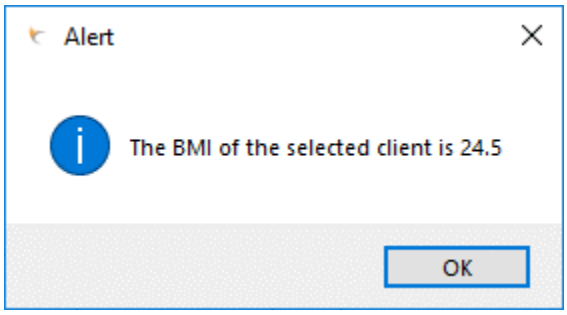
```
def delete_client(self):
    """
    deletes highlighted client
    """
    iid_selected = self.tree.focus()
    medical_record_id = self.tree.item(iid_selected, 'text')

    self.cursor.execute("""DELETE from medicalrecords WHERE MedicalRecordID = ? """, (medical_record_id,))
    self.cursor.execute("""DELETE from clients WHERE MedicalRecordID = ? """, (medical_record_id,))
    self.db.commit()
    self.update_table()
```

Candidate Number:
3097

6.25: Medical records BMI calculator

```
def calculate_bmi(self):
    """
    calculates bmi of selected client
    """
    iid_selected = self.tree.focus()
    medical_record_id = self.tree.item(iid_selected, 'text')
    self.cursor.execute("""SELECT * FROM medicalrecords WHERE MedicalRecordID = ?""", (medical_record_id,))
    fetch = self.cursor.fetchone()
    try:
        height = float(fetch[5])
        mass = float(fetch[6])
        height_squared = height * height
        bmi = mass/height_squared
        bmi = round(bmi, 1)
        messagebox.showinfo("Alert", "The BMI of the selected client is " + str(bmi))
        self.db.commit()
        self.update_table()
    except TypeError:
        messagebox.showinfo("Error", "Mass and/or height is None")
```

Which creates:



6.26: Medical records validation

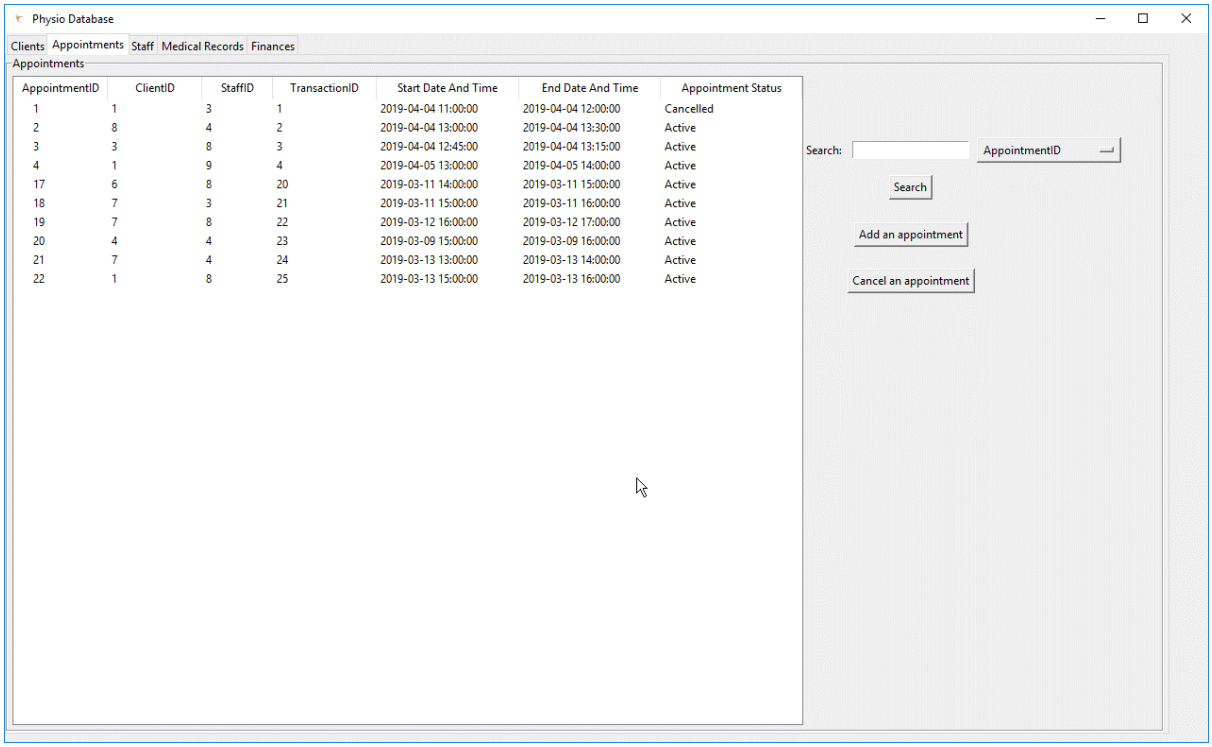
```
def add_medical_record(self):
    """
    checks if medical record results are valid and then updates tables
    """

    pattern = re.compile('\d+(\.\d+)?')

    medicalrecordvariable = self.medicalrecordvariable.get()
    medicalrecordvariable = (ast.literal_eval(medicalrecordvariable)[0])  # converts to tuple

    if self.sexvariable.get() == "":
        pass
    else:
        self.cursor.execute("""UPDATE medicalrecords SET Sex = ? WHERE MedicalRecordID = ?""",
                            (self.sexvariable.get(), medicalrecordvariable,))

    if self.gendervariable.get() == "":
        pass
    else:
        self.cursor.execute("""UPDATE medicalrecords SET Gender = ? WHERE MedicalRecordID = ?""",
                            (self.gendervariable.get(), medicalrecordvariable,))

    if self.bloodtypevariable.get() == "":
        pass
    else:
        self.cursor.execute("""UPDATE medicalrecords SET BloodType = ? WHERE MedicalRecordID = ?""",
                            (self.bloodtypevariable.get(), medicalrecordvariable,))

    match = re.search(pattern, self.height_box.get())
    if self.height_box.get() == "":
        pass
    elif match:
        if 1 < float(self.height_box.get()) < 2.5:
            self.cursor.execute("""UPDATE medicalrecords SET Height = ? WHERE MedicalRecordID = ?""",
                                (self.height_box.get(), medicalrecordvariable,))
        else:
            messagebox.showinfo("Error", "Height is an invalid amount")
    else:
        messagebox.showinfo("Error", "Height contains non numerical characters")


match = re.search(pattern, self.mass_box.get())
if self.mass_box.get() == "":
    pass
elif match:
    if 30 < float(self.mass_box.get()) < 500:
        self.cursor.execute("""UPDATE medicalrecords SET Mass = ? WHERE MedicalRecordID = ?""",
                            (self.mass_box.get(), medicalrecordvariable,))
    else:
        messagebox.showinfo("Error", "Mass is an invalid amount")
else:
    messagebox.showinfo("Error", "Mass contains non numerical characters")

self.db.commit()
self.edit_medical_record_window.destroy()
MedicalRecordsFrame.update_table(self)
```

Candidate Number:
3097

6.27: Appointments tab GUI

```
# coding=utf-8

import tkinter as Tkinter
import tkinter.ttk as ttk
import re
import sqlite3
from tkinter import *
import tkinter as tk
import datetime
from datetime import datetime
from tkinter import messagebox
import ast


class AppointmentsFrame(tk.Frame):
    """
    the frame for the appointments tab
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)
        appointments_frame = tk.LabelFrame(self, text="Appointments", padx=5, pady=5, width=1250, height=750)
        appointments_frame.grid(row=0, column=0)

        OPTIONS = [
            "AppointmentID          ",
            "ClientID                       ",
            "StaffID                         ",
            "TransactionID              ",
            "Start Date And Time ",
            "End Date And Time   ",
            "Appointment Status  "
        ]

        variable = StringVar(appointments_frame)
        variable.set(OPTIONS[0])

        # Set the treeview
        appointments_frame.tree = ttk.Treeview(appointments_frame, height="33", selectmode='browse',
                                               columns=(
                                                   'ClientID', 'StaffID', 'TransactionID', 'Start Date And Time',
                                                   'End Date And Time', 'Appointment Status'
                                               ))
        AppointmentsFrame.tree = appointments_frame.tree
        appointments_frame.tree.heading('#0', text='AppointmentID')
        appointments_frame.tree.heading('#1', text='ClientID')
        appointments_frame.tree.heading('#2', text='StaffID')
        appointments_frame.tree.heading('#3', text='TransactionID')
        appointments_frame.tree.heading('#4', text='Start Date And Time')
appointments_frame.tree.heading('#5', text='End Date And Time')
appointments_frame.tree.heading('#6', text='Appointment Status')
appointments_frame.tree.column('#0', stretch=Tkinter.YES, width="100", minwidth="50")
appointments_frame.tree.column('#1', stretch=Tkinter.YES, width="100", minwidth="100")
appointments_frame.tree.column('#2', stretch=Tkinter.YES, width="75", minwidth="50")
appointments_frame.tree.column('#3', stretch=Tkinter.YES, width="110", minwidth="85")
appointments_frame.tree.column('#4', stretch=Tkinter.YES, width="150", minwidth="85")
appointments_frame.tree.column('#5', stretch=Tkinter.YES, width="150", minwidth="75")
appointments_frame.tree.column('#6', stretch=Tkinter.YES, width="150", minwidth="100")
appointments_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
appointments_frame.treeview = appointments_frame.tree

search = tk.Label(appointments_frame, text="Search: ")
search.grid(row=10, column=50)
search_box = appointments_frame.search_entry = tk.Entry(appointments_frame)
search_box.grid(row=10, column=51)
dropdownsearch = OptionMenu(appointments_frame, variable, *OPTIONS)
dropdownsearch.grid(row=10, column=52)

tk.Button(appointments_frame, text="Search", command=self.search_table).grid(row=11, column=51)

tk.Button(appointments_frame, text="Add an appointment", command=CreateAppointmentFrame).grid(row=13, column=51)

tk.Button(appointments_frame, text="Cancel an appointment", command=self.cancel_appointment).grid(row=15,
                                                                                                   column=51)

pad = tk.Label(appointments_frame, text="")
pad.grid(row=10, column=55, padx=(0, 30))

self.tree = appointments_frame.tree
self.variable = variable
self.search_box = search_box
self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()
self.update_table()
```

Candidate Number:
3097



6.28: Update appointments table

```
def update_table(self):
    """
    updates the treeview and fills it with all records
    in the AppointmentID table
    """
    self.cursor.execute("""SELECT * FROM appointments""")
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:])
```

Candidate Number:
3097

6.29: Search appointments

```
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "AppointmentID          ":
        self.cursor.execute("""SELECT * FROM appointments WHERE AppointmentID LIKE ?""", ('%' + search + '%',))
        drop_down = "AppointmentID"
    elif drop_down == "ClientID                        ":
        self.cursor.execute("""SELECT * FROM appointments WHERE ClientID LIKE ?""", ('%' + search + '%',))
        drop_down = "ClientID"
    elif drop_down == "StaffID                          ":
        self.cursor.execute("""SELECT * FROM appointments WHERE StaffID LIKE ?""", ('%' + search + '%',))
        drop_down = "StaffID"
    elif drop_down == "TransactionID            ":
        self.cursor.execute("""SELECT * FROM appointments WHERE TransactionID LIKE ?""", ('%' + search + '%',))
        drop_down = "TransactionID"
    elif drop_down == "Start Date And Time ":
        self.cursor.execute("""SELECT * FROM appointments WHERE StartDateAndTime LIKE ?""", ('%' + search + '%',))
        drop_down = "Start Date And Time"
    elif drop_down == "End Date And Time   ":
        self.cursor.execute("""SELECT * FROM appointments WHERE EndDateAndTime LIKE ?""", ('%' + search + '%',))
        drop_down = "End Date And Time"
    elif drop_down == "Appointment Status  ":
        self.cursor.execute("""SELECT * FROM appointments WHERE AppointmentStatus LIKE ?""", ('%' + search + '%',))
        drop_down = "Appointment Status"
    self.tree.delete(*self.tree.get_children())
    rows = self.cursor.fetchall()
    f = open('appointments.txt','w')
    f.write("------------------------ Searched for '" + search + "' by " + drop_down + "------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "AppointmentID: " + str(row[0]))
        f.write('\n' + "ClientID: "  + str(row[1]))
        f.write('\n' + "StaffID: " + str(row[2]))
        f.write('\n' + "TransactionID: " + str(row[3]))
        f.write('\n' + "Start Date And Time: " + str(row[4]))
        f.write('\n' + "End Date And Time: " + str(row[5]))
        f.write('\n' + "Appointment Status: " + str(row[6]))
        f.write('\n')
    messagebox.showinfo("Alert", "Appointments saved (appointments.txt)")
    f.close()
```

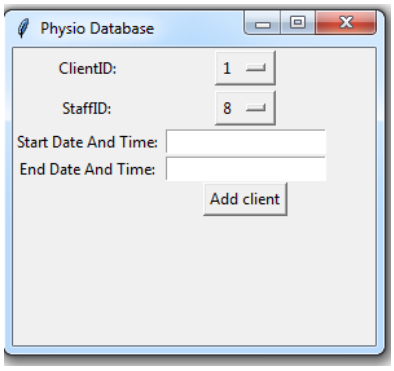
Which creates:

Candidate Number:
3097

6.30: Create appointments GUI

```
class CreateAppointmentFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new client
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = AppointmentsFrame.tree
        create_appointment_window = tk.Toplevel(self)
        create_appointment_window.geometry("280x230")
        self.create_appointment_window = create_appointment_window
        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        self.cursor.execute("""SELECT ClientID FROM clients""")

        CLIENTOPTIONS = []
        result = self.cursor.fetchall()
        for item in result:
            CLIENTOPTIONS.append(item)

        clientvariable = StringVar(create_appointment_window)

        try:
            clientvariable.set(CLIENTOPTIONS[0])
        except IndexError:
            messagebox.showinfo("Error", "No clients available")
            self.create_appointment_window.destroy()

        self.cursor.execute("""SELECT StaffID FROM staff""")

        STAFFOPTIONS = []
        result = self.cursor.fetchall()
        for item in result:
            STAFFOPTIONS.append(item)

        staffvariable = StringVar(create_appointment_window)

        try:
            staffvariable.set(STAFFOPTIONS[0])
        except IndexError:
            messagebox.showinfo("Error", "No staff available")
            self.create_appointment_window.destroy()
```

Which creates:




```
lientID: ")
client.grid(row=0, column=0)
clientsearch = OptionMenu(create_appointment_window, clientvariable, *CLIENTOPTIONS)
clientsearch.grid(row=0, column=1)

staff = tk.Label(create_appointment_window, text="StaffID: ")
staff.grid(row=1, column=0)
staffsearch = OptionMenu(create_appointment_window, staffvariable, *STAFFOPTIONS)
staffsearch.grid(row=1, column=1)

startdateandtime = tk.Label(create_appointment_window, text="Start Date And Time: ")
startdateandtime.grid(row=2, column=0)
startdateandtime_box = create_appointment_window.search_entry = tk.Entry(create_appointment_window)
startdateandtime_box.grid(row=2, column=1)

enddateandtime = tk.Label(create_appointment_window, text="End Date And Time: ")
enddateandtime.grid(row=3, column=0)
enddateandtime_box = create_appointment_window.search_entry = tk.Entry(create_appointment_window)
enddateandtime_box.grid(row=3, column=1)

search_button = tk.Button(create_appointment_window, text="Add client", command=self.add_appointment)
search_button.grid(row=4, column=1)

self.clientvariable = clientvariable
self.staffvariable = staffvariable
self.startdateandtime_box = startdateandtime_box
self.enddateandtime_box = enddateandtime_box

self.enddateandtime_box.bind("<Return>", self.add_appointment)
```

Candidate Number:
3097

```
------------------------ Searched for '' by AppointmentID------------------------

AppointmentID: 1
ClientID: 1
StaffID: 3
TransactionID: 1
Start Date And Time: 2019-04-04 11:00:00
End Date And Time: 2019-04-04 12:00:00
Appointment Status: Cancelled

AppointmentID: 2
ClientID: 8
StaffID: 4
TransactionID: 2
Start Date And Time: 2019-04-04 13:00:00
End Date And Time: 2019-04-04 13:30:00
Appointment Status: Active

AppointmentID: 3
ClientID: 3
StaffID: 8
TransactionID: 3
Start Date And Time: 2019-04-04 12:45:00
End Date And Time: 2019-04-04 13:15:00
Appointment Status: Active

AppointmentID: 4
ClientID: 1
StaffID: 9
TransactionID: 4
Start Date And Time: 2019-04-05 13:00:00
End Date And Time: 2019-04-05 14:00:00
Appointment Status: Active

AppointmentID: 17
ClientID: 6
StaffID: 8
TransactionID: 20
Start Date And Time: 2019-03-11 14:00:00
End Date And Time: 2019-03-11 15:00:00
Appointment Status: Active

AppointmentID: 18
ClientID: 7
StaffID: 3
TransactionID: 21
Start Date And Time: 2019-03-11 15:00:00
End Date And Time: 2019-03-11 16:00:00
Appointment Status: Active
```

Candidate Number:
3097

6.31: Cancel appointments

```
def cancel_appointment(self):
    """
    cancels highlighted appointment
    """
    iid_selected = self.tree.focus()
    appointment_id = self.tree.item(iid_selected, 'text')
    self.cursor.execute("""UPDATE appointments SET AppointmentStatus = ? WHERE AppointmentID = ?""",
                        ('Cancelled', appointment_id,))
    self.db.commit()
    self.update_table()
```

6.32: Appointments validation

```
def add_appointment(self):
    """
    checks if all appointment results are valid and then updates tables
    """
    clientvariable = self.clientvariable.get()
    clientvariable = (ast.literal_eval(clientvariable)[0])
    staffvariable = self.staffvariable.get()
    staffvariable = (ast.literal_eval(staffvariable)[0])
    try:
        datetime.strptime(self.startdateandtime_box.get(), '%Y-%m-%d %H:%M:%S')
        try:
            datetime.strptime(self.enddateandtime_box.get(), '%Y-%m-%d %H:%M:%S')
            self.cursor.execute(
                """INSERT INTO appointments(ClientID, StaffID, StartDateAndTime, EndDateAndTime, AppointmentStatus) VALUES (?,?,?,?,?)""",
                (clientvariable, staffvariable, self.startdateandtime_box.get(),
                 self.enddateandtime_box.get(), 'Active'))
            self.cursor.execute("""SELECT AppointmentID FROM appointments ORDER BY AppointmentID DESC LIMIT 1""")
            appointmentid = self.cursor.fetchone()
            appointmentid = appointmentid[0]
            now = datetime.now()
            now = now.strftime('%Y-%m-%d %H:%M:%S')
            self.cursor.execute("""INSERT INTO transactions(DateAndTime, TransactionStatus) VALUES (?,?)""", (now, "Sucessful"))
            self.cursor.execute(
                """SELECT TransactionID FROM transactions ORDER BY TransactionID DESC LIMIT 1""")
            transactionid = self.cursor.fetchone()
            transactionid = transactionid[0]
            self.cursor.execute("""UPDATE appointments SET TransactionID = ? WHERE AppointmentID = ?""",
                                (appointmentid, transactionid,))
            self.db.commit()
            self.create_appointment_window.destroy()
            AppointmentsFrame.update_table(self)
        except ValueError:
            messagebox.showinfo("Error", "End date and time incorrect format (YYYY-MM-DD HH:MM:SS)")
    except ValueError:
        messagebox.showinfo("Error", "Start date and time incorrect format (YYYY-MM-DD HH:MM:SS)")
```

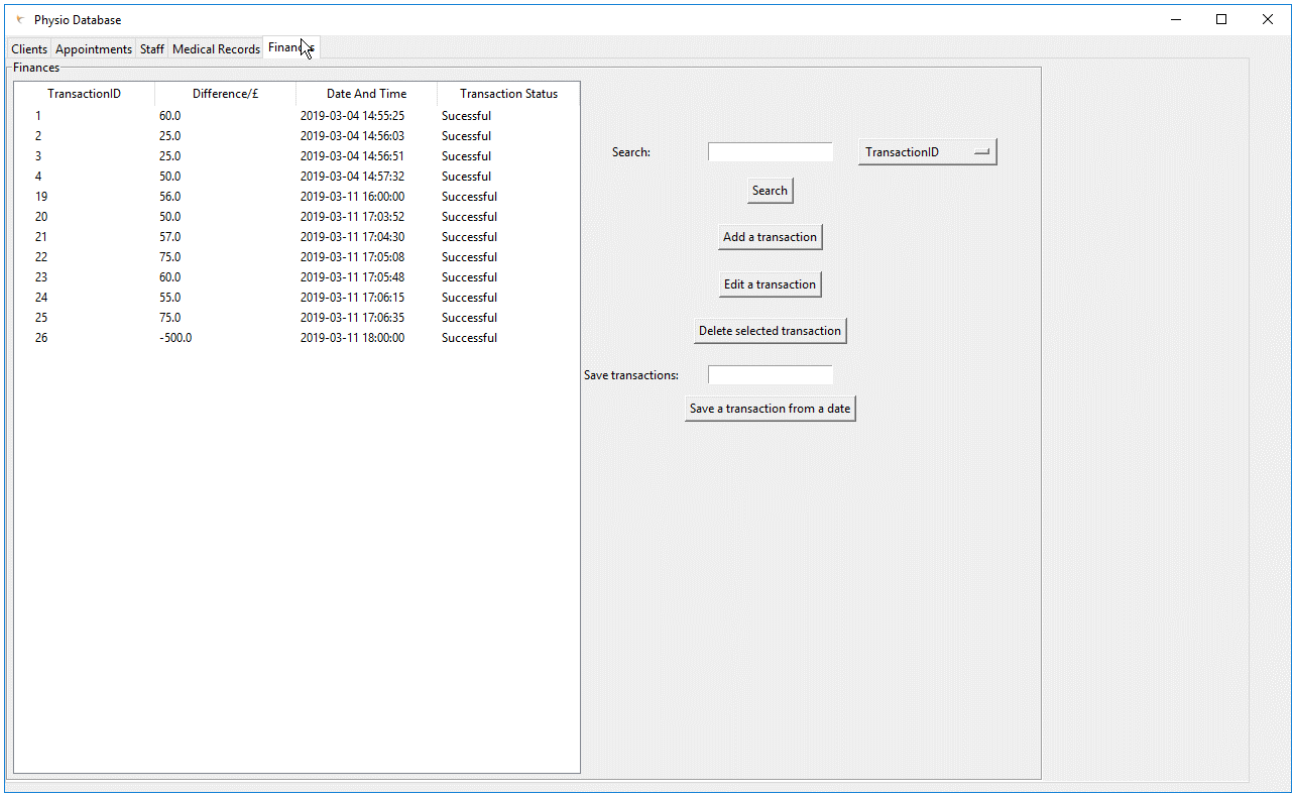
Candidate Number:
3097

6.33: Finance tab GUI

```
# coding=utf-8

import tkinter as Tkinter
import tkinter.ttk as ttk
import re
import sqlite3
from tkinter import *
import tkinter as tk
from datetime import datetime
from tkinter import messagebox
import ast


class FinancesFrame(tk.Frame):
    """
    the frame for the finances tab
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)
        finances_frame = tk.LabelFrame(self, text="Finances", padx=5, pady=5, width=1250, height=750)
        finances_frame.grid(row=0, column=0)

        OPTIONS = [
            "TransactionID        ",
            "Difference              ",
            "Date And Time      ",
            "Transaction Status"
        ]

        variable = StringVar(finances_frame)
        variable.set(OPTIONS[0])

        # Set the treeview
        finances_frame.tree = ttk.Treeview(finances_frame, height="33", selectmode='browse',
                                           columns=(
                                               'Difference', 'Date And Time', 'Transaction Status'))
        FinancesFrame.tree = finances_frame.tree
        finances_frame.tree.heading('#0', text='TransactionID')
        finances_frame.tree.heading('#1', text='Difference/£')
        finances_frame.tree.heading('#2', text='Date And Time')
        finances_frame.tree.heading('#3', text='Transaction Status')
        finances_frame.tree.column('#0', stretch=Tkinter.YES, width="140", minwidth="140")
        finances_frame.tree.column('#1', stretch=Tkinter.YES, width="140", minwidth="140")
        finances_frame.tree.column('#2', stretch=Tkinter.YES, width="140", minwidth="140")
        finances_frame.tree.column('#3', stretch=Tkinter.YES, width="140", minwidth="140")
        finances_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
        finances_frame.treeview = finances_frame.tree
```

Candidate Number:
3097

```
FinancesFrame.tree = finances_frame.tree
finances_frame.tree.heading('#0', text='TransactionID')
finances_frame.tree.heading('#1', text='Difference/£')
finances_frame.tree.heading('#2', text='Date And Time')
finances_frame.tree.heading('#3', text='Transaction Status')
finances_frame.tree.column('#0', stretch=Tkinter.YES, width="140", minwidth="140")
finances_frame.tree.column('#1', stretch=Tkinter.YES, width="140", minwidth="140")
finances_frame.tree.column('#2', stretch=Tkinter.YES, width="140", minwidth="140")
finances_frame.tree.column('#3', stretch=Tkinter.YES, width="140", minwidth="140")
finances_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
finances_frame.treeview = finances_frame.tree

search = tk.Label(finances_frame, text="Search: ")
search.grid(row=10, column=50)
search_box = finances_frame.search_entry = tk.Entry(finances_frame)
search_box.grid(row=10, column=51)
dropdownsearch = OptionMenu(finances_frame, variable, *OPTIONS)
dropdownsearch.grid(row=10, column=52)

save = tk.Label(finances_frame, text="Save transactions: ")
save.grid(row=19, column=50)
save_box = finances_frame.search_entry = tk.Entry(finances_frame)
save_box.grid(row=19, column=51)

tk.Button(finances_frame, text="Search", command=self.search_table).grid(row=11, column=51)

tk.Button(finances_frame, text="Add a transaction", command=CreateFinancesFrame).grid(row=13, column=51)

tk.Button(finances_frame, text="Edit a transaction", command=EditFinancesFrame).grid(row=15, column=51)

tk.Button(finances_frame, text="Delete selected transaction", command=self.delete_transaction).grid(row=17, column=51)

tk.Button(finances_frame, text="Save a transaction from a date", command=self.save_transaction).grid(row=20, column=51)

pad = tk.Label(finances_frame, text="")
pad.grid(row=10, column=55, padx=(0, 30))

self.tree = finances_frame.tree
self.variable = variable
self.search_box = search_box
self.save_box = save_box
self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()
self.update_table()
```



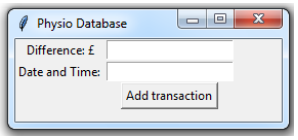
6.34: Update finance table

```
def update_table(self):
    """
    updates the treeview and fills it with all records
    in the transactions table
    """
    self.cursor.execute("""SELECT * FROM transactions""")
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:])
```

6.35: Create transaction GUI

```
class CreateFinancesFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new client,
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = FinancesFrame.tree
        create_transaction_window = tk.Toplevel(self)
        create_transaction_window.geometry("260x80")

        difference = tk.Label(create_transaction_window, text="Difference: £")
        difference.grid(row=0, column=0)
        difference_box = create_transaction_window.search_entry = tk.Entry(create_transaction_window)
        difference_box.grid(row=0, column=1)

        dateandtime = tk.Label(create_transaction_window, text="Date and Time: ")
        dateandtime.grid(row=1, column=0)
        dateandtime_box = create_transaction_window.search_entry = tk.Entry(create_transaction_window)
        dateandtime_box.grid(row=1, column=1)

        search_button = tk.Button(create_transaction_window, text="Add transaction", command=self.add_transaction)
        search_button.grid(row=2, column=1)

        self.difference_box = difference_box
        self.dateandtime_box = dateandtime_box
        self.create_transaction_window = create_transaction_window
        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        self.dateandtime_box.bind("<Return>", self.add_transaction)
```

Which creates:



Candidate Number:
3097

6.36: Search transactions

```
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "TransactionID        ":
        self.cursor.execute("""SELECT * FROM transactions WHERE TransactionID LIKE ?""", ('%' + search + '%',))
        drop_down = "TransactionID"
    elif drop_down == "Difference               ":
        self.cursor.execute("""SELECT * FROM transactions WHERE Difference LIKE ?""", ('%' + search + '%',))
        drop_down = "Difference"
    elif drop_down == "Date And Time      ":
        self.cursor.execute("""SELECT * FROM transactions WHERE DateAndTime LIKE ?""", ('%' + search + '%',))
        drop_down = "Date and Time"
    elif drop_down == "Transaction Status":
        self.cursor.execute("""SELECT * FROM transactions WHERE TransactionStatus LIKE ?""", ('%' + search + '%',))
    self.tree.delete(*self.tree.get_children())
    rows = self.cursor.fetchall()
    f = open('finances.txt','w')
    f.write("------------------------ Searched for '" + search + "' by " + drop_down + "------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "TransactionID: " + str(row[0]))
        f.write('\n' + "Difference: "  + str(row[1]))
        f.write('\n' + "Date and Time: " + str(row[2]))
        f.write('\n' + "Transaction Status: " + str(row[3]))
        f.write('\n')
    messagebox.showinfo("Alert", "Transaction saved (finances.txt)")
    f.close()
```

Which creates:

```
----------------------- Searched for '' by TransactionID-----------------------

TransactionID: 1
Difference: 60.0
Date and Time: 2019-03-04 14:55:25
Transaction Status: Sucessful

TransactionID: 2
Difference: 25.0
Date and Time: 2019-03-04 14:56:03
Transaction Status: Sucessful

TransactionID: 3
Difference: 25.0
Date and Time: 2019-03-04 14:56:51
Transaction Status: Sucessful

TransactionID: 4
Difference: 50.0
Date and Time: 2019-03-04 14:57:32
Transaction Status: Sucessful

TransactionID: 19
Difference: 56.0
Date and Time: 2019-03-11 16:00:00
Transaction Status: Successful

TransactionID: 20
Difference: 50.0
Date and Time: 2019-03-11 17:03:52
Transaction Status: Successful

TransactionID: 21
Difference: 57.0
Date and Time: 2019-03-11 17:04:30
Transaction Status: Successful

TransactionID: 22
Difference: 75.0
Date and Time: 2019-03-11 17:05:08
Transaction Status: Successful

TransactionID: 23
Difference: 60.0
Date and Time: 2019-03-11 17:05:48
Transaction Status: Successful

TransactionID: 24
Difference: 55.0
Date and Time: 2019-03-11 17:06:15
Transaction Status: Successful

TransactionID: 25
Difference: 75.0
```

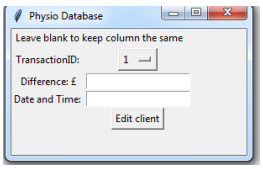
Candidate Number:
3097

6.37: Edit transaction GUI

```
class EditFinancesFrame(tk.Frame):
    """
    creates a new window and allows the user to edit a current client,
    when completed it will update the treeview in TransactionTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = FinancesFrame.tree
        edit_transaction_window = tk.Toplevel(self)
        edit_transaction_window.geometry("280x150")

        self.db = sqlite3.connect('clinic.db')
        self.cursor = self.db.cursor()

        self.cursor.execute("""SELECT TransactionID FROM transactions""")

        TRANSACTIONOPTIONS = []
        result = self.cursor.fetchall()
        for item in result:
            TRANSACTIONOPTIONS.append(item)

        transactionvariable = StringVar(edit_transaction_window)
        transactionvariable.set(TRANSACTIONOPTIONS[0])

        psa = tk.Label(edit_transaction_window, text="Leave blank to keep column the same ")
        psa.grid(row=0, column=0, columnspan=2)

        clientid = tk.Label(edit_transaction_window, text="TransactionID: ")
        clientid.grid(row=1, column=0)
        clientidsearch = OptionMenu(edit_transaction_window, transactionvariable, *TRANSACTIONOPTIONS)
        clientidsearch.grid(row=1, column=1)

        difference = tk.Label(edit_transaction_window, text="Difference: £")
        difference.grid(row=2, column=0)
        difference_box = edit_transaction_window.search_entry = tk.Entry(edit_transaction_window)
        difference_box.grid(row=2, column=1)

        dateandtime = tk.Label(edit_transaction_window, text="Date and Time: ")
        dateandtime.grid(row=3, column=0)
        dateandtime_box = edit_transaction_window.search_entry = tk.Entry(edit_transaction_window)
        dateandtime_box.grid(row=3, column=1)


        search_button = tk.Button(edit_transaction_window, text="Edit client", command=self.add_transaction)
        search_button.grid(row=4, column=1)
```

```
self.transactionvariable = transactionvariable
self.difference_box = difference_box
self.dateandtime_box = dateandtime_box
self.edit_transaction_window = edit_transaction_window
self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()

self.dateandtime_box.bind("<Return>", self.add_transaction)
```

Which creates:



6.38: Delete transaction

```
def delete_transaction(self):
    """
    deletes highlighted transaction
    """
    iid_selected = self.tree.focus()
    transaction_id = self.tree.item(iid_selected, 'text')

    self.cursor.execute("""DELETE from transactions WHERE TransactionID = ? """, (transaction_id,))
    self.db.commit()
    self.update_table()
```

Candidate Number:
3097

6.39: Transaction validation

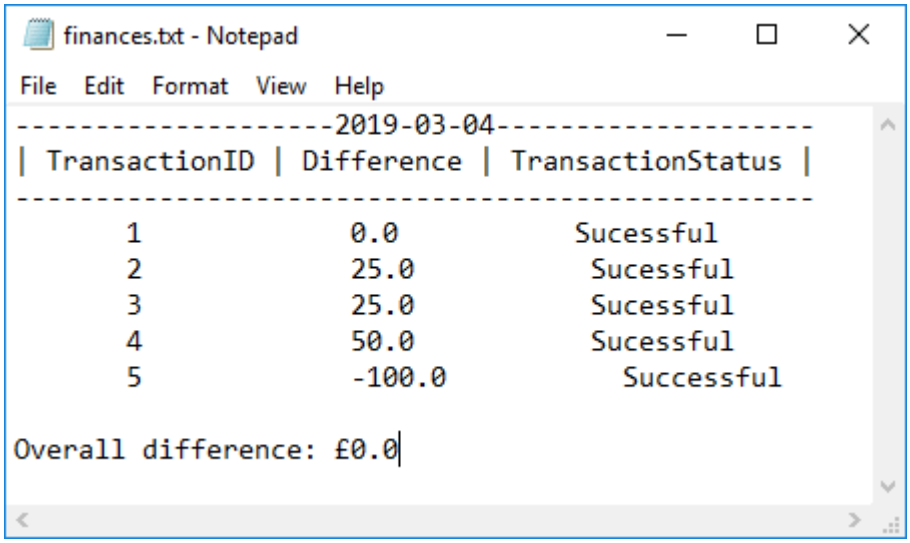
```
def add_transaction(self):
    """
    checks if add client results are valid and then updates tables
    """
    pattern = re.compile('\d+(\.\d+)?')
    match = re.search(pattern, self.difference_box.get())
    if match:
        try:
            datetime.strptime(self.dateandtime_box.get(), '%Y-%m-%d %H:%M:%S')
            self.cursor.execute(
                                """INSERT INTO transactions(Difference, DateAndTime, TransactionStatus) VALUES (?,?,?)""",
                                (self.difference_box.get(), self.dateandtime_box.get(), "Successful",))
            self.db.commit()
            self.create_transaction_window.destroy()
            FinancesFrame.update_table(self)
        except ValueError:
            messagebox.showinfo("Error", "Date and time incorrect format (YYYY-MM-DD HH:MM:SS)")
    else:
        messagebox.showinfo("Error", "Difference is invalid format")


def add_transaction(self):
    """
    checks if all transaction results are valid and then updates tables
    """
    transactionvariable = self.transactionvariable.get()
    transactionvariable = (ast.literal_eval(transactionvariable)[0])  # converts to tuple
    pattern = re.compile('\d+(\.\d+)?')
    match = re.search(pattern, self.difference_box.get())

    if self.difference_box.get() == "":
        pass
    else:
        if match:
            self.cursor.execute("""UPDATE transactions SET Difference = ? WHERE TransactionID = ?""",
                                (self.difference_box.get(), transactionvariable,))
        else:
            messagebox.showinfo("Error", "Transaction incorrect format (+/-DD)")

    if self.dateandtime_box.get() == "":
        pass
    else:
        try:
            datetime.strptime(self.dateandtime_box.get(), '%Y-%m-%d %H:%M:%S')
            self.cursor.execute("""UPDATE transactions SET DateAndTime = ? WHERE TransactionID = ?""",
                                (self.dateandtime_box.get(), transactionvariable,))
        except ValueError:
            messagebox.showinfo("Error", "Date and time incorrect format (YYYY-MM-DD HH:MM:SS)")

    self.db.commit()
    self.edit_transaction_window.destroy()
    FinancesFrame.update_table(self)
```

Candidate Number:
3097

6.40: Transaction receipt output

```
def save_transaction(self):
    date = self.save_box.get()
    pattern = r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'
    match = re.search(pattern, date)
    if match:
        self.cursor.execute("""SELECT * FROM transactions WHERE DateAndTime LIKE ?""", ('%' + date + '%',))
        rows = self.cursor.fetchall()
        self.cursor.execute("""SELECT sum(Difference) FROM transactions WHERE DateAndTime LIKE ?""", ('%' + date + '%',))
        sum = self.cursor.fetchone()
        f = open('finances.txt','w')
        f.write("--------------------" + date + "--------------------" + '\n')
        f.write("| TransactionID | Difference | TransactionStatus |"  + '\n')
        f.write("----------------------------------------------------")
        for row in rows:
            f.write('\n' + "       " + str(row[0]) + "             "  + str(row[1]) + "           " + row[3])
        f.write('\n' + '\n' + "Overall difference: £" + str(sum[0]))
        f.close()
        messagebox.showinfo("Alert", "Receipt added")
    else:
        messagebox.showinfo("Error", "Date incorrect format (YYYY-MM-DD)")
```

Which creates:



6.41: Comparison to the specification

Taken full account of the revised detailed design and refined the prototype in light of feedback and changed designs

Comparing my design to the software development, it is clear that when comparing between both of them, the program is very close to everything that I planned to do within the Design stage. You can see that the tables, GUI, algorithms and data structures are matched closely to it. You can also see that I have added the appropriate sections to the prototype based on the feedback in the post-prototype. For example, I added validation and the remaining tables

Produced a functioning solution to a highly demanding problem that meets most of the objectives for the solution to the problem. Better solutions will meet almost all of the objectives for the proposed solution.

I feel like the scope that I have used for my program which I laid out in my discussion was suitable and to a very high standard in addition to being demanding. Comparing to the success criteria. Green = successful.

Clients

- The program must allow the user to add a client to the clients table if all information added is valid
- The program must validate all data inputted into the clients table
- The program must list all clients in a table
- The program must allow the user to search for a clients of a specific keyword in a specific column and output it into the program and into a text file
- The program must allow a user to add a client and create a medical record connected to that client's primary key at the same time
- The program must allow a user to edit a client
- The program must allow a user to delete a client
- The program must list all client information in a clear fashion

Medical records

- The program must validate all data inputted into the medical records table
- The program must list all medical records in a table
- The program must allow the user to search for a medical records of a specific keyword in a specific column and output it into the program and into a text file
- The program must automatically create a medical record when a respective client is added and connect to that client
- The program must allow a user to edit a medical record
- The program must allow a user to delete a medical record
- The program must list all medical record information in a clear fashion
- The program must allow a user to calculate the BMI of a selected medical record

Candidate Number:
3097

Appointments

- The program must allow the user to add an appointment to the appointments table if all information added is valid
- The program must validate all data inputted into the appointments table
- The program must list all appointments in a table
- The program must allow the user to search for an appointment of a specific keyword in a specific column and output it into the program and into a text file
- The program must allow a user to add an appointment and create a transaction connected to that appointment's primary key at the same time in addition to saving the date and time of that transaction
- The program must allow a user to cancel an appointment
- The program must list all appointment information in a clear fashion

Transactions

- The program must allow the user to add a transaction to the transactions table if all information added is valid
- The program must validate all data inputted into the transactions table
- The program must list all transactions in a table
- The program must allow the user to search for a transactions of a specific keyword in a specific column and output it into the program and into a text file
- The program must allow a user to add a transaction
- The program must allow a user to edit a transaction
- The program must allow a user to delete a transaction
- The program must allow a user to output all transactions of a particular date and output the total sum of that date
- The program must list all transaction information in a clear fashion

Staff

- The program must allow the user to add a staff to the staff table if all information added is valid
- The program must validate all data inputted into the staff table
- The program must list all staffs in a table
- The program must allow the user to search for a staff of a specific keyword in a specific column and output it into the program and into a text file
- The program must allow a user to add a staff member
- The program must allow a user to edit a staff member
- The program must allow a user to delete a staff member
- The program must list all staff information in a clear fashion
- Usernames and passwords should be stored in this table however passwords will not be able to be seen within the program

Candidate Number:
3097

Overall

- The program must be in third normalised form
- The database must be stored in SQL
- The program must implement views
- The program must contain a GUI
- The program must be fast
- The program must contain a readme.txt to guide any new users on how to use the program and how to troubleshoot
- The program must be intuitive and easy to understand, and it shouldn't be difficult to use any of the functions

Used and fully exploited the programming facilities of the language

I have used a very wide range of the programming facilities in Python and the code that I have written in my program is complex. You can see examples of this, for example the use of classes within my software development at 6.2, I have also used suitable tools such as regular expression for format checks which allow me to look for all possible formats of a given input. You can see this in my postcode check.

Demonstrated a sound understanding of the appropriate tools and techniques available to them

If you look through the code that I have created, it is clear that I have chosen the appropriate methods to create functions, for example for displaying data I did a wide amount of research and found that the treeview was the most appropriate method. You can see examples of this when I do things such as creating functions that delete a record. You can see this at 6.11

Created a well-structured data model that aids efficient data handling. Better candidates will present a data model normalised to third normal form.

My data structure is fast and efficient and relational. It is will structured with suitable data types selected for each area. I have normalised my program to third normalised form, you can see evidence of this in the Design stage.

Produced a solution that is well-structured and modular in nature. The solution makes good use of local variables and minimises the use of global variables.

The solution uses a variety of classes and functions and uses no global variables within it. If you look throughout my code and CTRL + F, you will notice no occurrences of global variables come up. You can also see that there are separate python files for each tab making the program much more modular.

Candidate Number:
3097

Written code that is fully self-documenting and well-structured with annotation that will allow a competent third party to maintain the solution in future

A range of comments has been used throughout the program in addition to the program using self documenting identifiers and suitable variable and class names for users to understand. I have shown some of my classmates my code and they seemed to understand a significant amount of it.

Made use of complex user-defined routines. Better candidates will have made effective use of recursive algorithms.

The program uses a significant number of recursive algorithms, for example you can see this for the line in lines when outputting the table and at 6.10, the use of these recursive algorithms make the program significantly more efficient and faster

Produced evidence of the effective use of validation for all key components. Better candidates will have created efficient routines for exception handling.

See the validation sections of the program, a wide range of validation has been used and some exception handling has been used to, for example if a user tries to edit records when no records exist it will produce an error, this is through a try-except check. You can see an example of this in 6.12

Fully documented the variables and actual data structures used to create the solution to the chosen problem

You can see the structure of the data in the design stage. The documentation of variables is below:

Login frame

Variable name	Description	Data type	Example	Validation
new_frame	The frame the program will be changed to	String	TabFrame	None
self.un_entry	The entry box of the username	String	-	None
self.pw_entry	The entry box of the password	String	-	None
username	The value of the username	String	owneruser	None
password	The value of the password	String	ownerpass	None
result	Checks if the username + password search gets a result	Boolean	True	None
pos	Gets the position of the user that is logging in to give them a suitable position	String	Owner	Lookup check

Candidate Number:
3097

Client frame

Variable name	Description	Data type	Example	Validation
clients_frame	The description and attributes of the client's frame	String	tk.LabelFrame(self, text="Clients", padx=5, pady=5, width=1275, height=750)	None
self.variable	The drop-down list in search	String	StringVar(clients_frame)	None
self.db	Connects the database to the clinic	String	-	None
self.cursor	Sets out the cursor	String	sqlite3.connect('clinic.db')	None
Iid_selected	Gets selected iid of highlighted box	String	I00A	None
pos	Gets the position of the user that is logging and shows them suitable buttons	String	Owner	Lookup check
client_id	Gets the client id of the iid	Integer	1	None
drop_down	gets the drop down of variable	String	'ClientID'	Lookup check
rows	Gets all records in the SQL file	String	[(1, 1, 'Mr', 'Kai', 'Holloway', '1934-07-11', '07863701077', '23 Ermin Street', 'RH18 9WX')]	None
f	File chooses the file name and how it is opened	String	open('clients.txt','w')	None
prefixoptions	The dropdown values for prefix	Array	["", "Dr", "Mr", "Mrs", "Ms", "Mx", "Prof", "Rev"]	None
dropdownsearch	The prefix dropdown value	String	Dr	Presence
first_name_box	The first name box	String	Dayyan	Presence, length, alphabetical

surname_box	The surname box	String	OBrien	Presence, length, alphabetical
dob_box	The DOB box	Date	31-01-2001	Presence, format
telephone_box	The telephone box	String	07767862981	Presence, length, numerical
address_box	The address box	String	10 Windsor Hill	Presence, length
postcode_box	The postcode box	String	BT34 1ER	Presence, format
search_button	The search button	String	tk.Button(edit_client_window, text="Edit client", command=self.add_client)	None
match	Chooses what pattern is matched with which value	String	re.search(pattern, self.dob_box.get())	None
pattern	Creates the pattern to be matched to	String	r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'	None
CLIENTOPTIONS	The array of all existing clients	Array	[]	None
clientidsearch	The selected client	Integer	1	None

Candidate Number:
3097

Staff frame

Variable name	Description	Data type	Example	Validation
staff_frame	The description and attributes of the staff's frame	String	tk.LabelFrame(self, text="Staff", padx=5, pady=5, width=1275, height=750)	None
self.variable	The drop-down list in search	String	StringVar(staff_frame)	None
self.db	Connects the database to the clinic	String	-	None
self.cursor	Sets out the cursor	String	sqlite3.connect('clinic.db')	None
Iid_selected	Gets selected iid of highlighted box	String	I00A	None
drop_down	gets the drop down of variable	String	'StaffID'	Lookup check
rows	Gets all records in the SQL file	String	[(1, 'Dr', 'Kirsten', 'Frener', '2001-01-01', '07901611233', '10 Windsor Hill', 'BT341ER', 'owneruser', 'Owner', 1, 'ownerpass')]	None
f	File, chooses the file name and how it is opened	String	open('clients.txt','w')	None
prefixoptions	The dropdown values for prefix	Array	["", "Dr", "Mr", "Mrs", "Ms", "Mx", "Prof", "Rev"]	None
dropdownsearch	The prefix dropdown value	String	Dr	Presence
positionoptions	The dropdown values for the position	Array	["Owner", "IT", "Nurse", "Physiotherapist", "Receptionist"]	None

posistiondropdownsearch	The position dropdown value	String	Owner	Presence
first_name_box	The first name box	String	Dayyan	Presence, length, alphabetical
surname_box	The surname box	String	OBrien	Presence, length, alphabetical
dob_box	The DOB box	Date	31-01-2001	Presence, format
telephone_box	The telephone box	String	07767862981	Presence, length, numerical
address_box	The address box	String	10 Windsor Hill	Presence, length
postcode_box	The postcode box	String	BT34 1ER	Presence, format
username_box	The username box	String	owneruser	Presence, length
password_box	The password box	String	ownerpass	Presence, length
search_button	The search button	String	tk.Button(create_staff_window, text="Add staff", command=self.add_staff)	None
match	Chooses what pattern is matched with which value	String	re.search(pattern, self.dob_box.get())	None
pattern	Creates the pattern to be matched to	String	r'(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])'	None
STAFFOPTIONS	The array of all existing staff	Array	[]	None
staffidsearch	The selected staff	Integer	1	None

Candidate Number:
3097

Transaction frame

Variable name	Description	Data type	Example	Validation
finances_frame	The description and attributes of the finances frame	String	finances_frame = tk.LabelFrame(self, text="Finances", padx=5, pady=5, width=1250, height=750)	None
self.variable	The drop-down list in search	String	variable = StringVar(finances_frame)	None
self.db	Connects the database to the clinic	String	-	None
self.cursor	Sets out the cursor	String	sqlite3.connect('clinic.db')	None
Iid_selected	Gets selected iid of highlighted box	String	I00A	None
transaction_id	Gets the transaction id of the iid	Integer	1	None
drop_down	gets the drop down of variable	String	'TransactionID'	Lookup check
rows	Gets all records in the SQL file	String	[(1, 60.0, '2019-03-04 14:55:25', 'Successful')]	None
f	File, chooses the file name and how it is opened	String	open('clients.txt','w')	None
difference_box	The difference box	String	Dayyan	Presence, format
dateandtime_box	The date and time	String	OBrien	Presence, format
search_button	The search button	String	tk.Button(create_transaction_window, text="Add transaction", command=self.add_transaction)	None

match	Chooses what pattern is matched with which value	String	re.search(pattern, self.difference_box.get())	None
pattern	Creates the pattern to be matched to	String	re.compile('\d+(\.\d+)?')	None
TRANSACTIONOPTIONS	The array of all existing transaction	Array	[]	None
transactionidsearch	The selected transaction	Integer	1	None
date	The date to save transaction for	Date	2019-03-11	Presence, format
sum	The total sum of all transaction of a date	Real	59.43	None

Candidate Number:
3097

Medical records fame

Variable name	Description	Data type	Example	Validation
medical_records_frame	The description and attributes of the client's frame	String	tk.LabelFrame(self, text="Medical Records", padx=5, pady=5, width=1000, height=750)	None
self.variable	The drop-down list in search	String	StringVar(clients_frame)	None
self.db	Connects the database to the clinic	String	-	None
self.cursor	Sets out the cursor	String	sqlite3.connect('clinic.db')	None
Iid_selected	Gets selected iid of highlighted box	String	I00A	None
medical_record_id	Gets the client id of the iid	Integer	1	None
drop_down	Gets the drop down of variable	String	'MedicalRecordsID'	Lookup check
rows	Gets all records in the SQL file	String	[(1, 1, 'M', 'Male', 'B-', 6.75, 75.0)]	None
f	File chooses the file name and how it is opened	String	open('medicalrecords.txt','w')	None
SEXOPTIONS	The dropdown values for prefix	Array	["", "M", "F"]	None
sexsearch	The prefix dropdown value	String	M	Presence
GENDEROPTIONS	The dropdown	Array	["", "Male", "Female"]	None

	values for prefix			
gendersearch	The prefix dropdown value	String	Male	Presence
BLOODTYPEOPTIONS	The dropdown values for prefix	Array	["", "A+”, “A-“, “B+”, “B-“, “O+”, “O-“, “AB+”, “AB-“]	None
bloodtypesearch	The prefix dropdown value	String	A+	Presence
height	The height box	String	6.70	Presence, range
mass	The mass box	String	70	Presence, range
search_button	The search button	String	tk.Button(edit_medical_record_window, text="Edit Medical Record", command=self.add_medical_record)	None
MEDICALRECORDOPTIONS	The array of all existing clients	Array	[]	None
medicalrecordsidsearch	The selected client	Integer	1	None

Candidate Number:
3097

Appointments frame

Variable name	Description	Data type	Example	Validation
appointments_frame	The description and attributes of the appointment's frame	String	tk.LabelFrame(self, text="Appointment's", padx=5, pady=5, width=1275, height=750)	None
self.variable	The drop-down list in search	String	StringVar(appointment_frame)	None
self.db	Connects the database to the clinic	String	-	None
self.cursor	Sets out the cursor	String	sqlite3.connect('clinic.db')	None
Iid_selected	Gets selected iid of highlighted box	String	I00A	None
appointment_id	Gets the appointment id of the iid	Integer	1	None
drop_down	gets the drop down of variable	String	'AppointmentID'	Lookup check
rows	Gets all records in the SQL file	String	[(1, 1, 3, 1, '2019-04-04 11:00:00', '2019-04-04 12:00:00', 'Cancelled')]	None
f	File chooses the file name and how it is opened	String	open('appointments.txt','w')	None
STAFFOPTIONS	The array of all existing staff	Array	[]	None
CLIENTOPTIONS	The array of all existing clients	Array	[]	None
startdateandtime_box	The start date and time box	String	2019-03-11 15:00:00	Presence, format
enddateandtime_box	The end date and time box	String	2019-03-11 16:00:00	Presence, format
now	Current date and time	String	2019-03-11 17:30:00	None
search_button	The search button	String	tk.Button(edit_client_window, text="Edit client", command=self.add_client)	None

Candidate Number:
3097

Database

Variable name	Description	Data type	Example	Validation
db	Connects the program to the database	String	sqlite3.connect('clinic.db')	None
cursor	Creates the cursor	String	db.cursor()	None

Included evidence of the completed user interface including a full description of the features that make it fit for audience and purpose.

You can see suitable evidence for the user interface throughout the software development, in sections such as 6.2 and 6.6. The description of features are below:

- The program uses a GUI instead of being a text-based CLI, this makes it much easier for staff members to use and understand
- The vast majority of the program is in one frame and thus allows the user to easily navigate and flow throughout the program
- There are a wide range of outputs, some in the treeview, some in text files and some as alerts
- The system allows you to easily, add, edit, delete and search for records
- All tables within the program are separated by tabs
- The layout of the program is consistent
- The program is normalised to 3NF and fully relational
- A large range of validation is used throughout the program