Implementation

Main

```
1.  #Gymnasium Database Management System
2.
3.  import sys
4.
5.  from gym_delete_dialog_class import *
6.  from gym_print_dialog_class import *
7.  from gym_search_dialog_class import *
8.  from gym_edit_dialog_class import *
9.  from gym_add_edit_dialog_class import *
10. from gym_add_dialog_class import *
11. from gym_about_dialog_class import *
12. from gym_password_dialog_class import *
13.
14. from data_browser import *
15.
16. from PyQt4.QtCore import *
17. from PyQt4.QtGui import *
18.
19. class AppWindow(QMainWindow):
20.     """creates the main window"""
21.
22.     #constructor
23.     def __init__(self):
24.         super().__init__()
25.         self.setWindowTitle("Gym Database Management System 9001")#sets the title for the
    window
26.         self.setWindowIcon(QIcon("Logo.png"))#sets the window icon
27.
28.         #variable for database
```

```
29.         self.file = None
30.
31.         #toolbars
32.         self.open_push_button = QPushButton("Open")#sets the main button for opening the Open
    GUI and function
33.         self.add_push_button = QPushButton("Add")#sets the main button for opening the Add GUI
    and function
34.         self.edit_push_button = QPushButton("Edit")#sets the main button for opening the Edit GUI
    and function
35.         self.delete_push_button = QPushButton("Delete")#sets the main button for opening the
    Delete GUI and function
36.         self.search_push_button = QPushButton("Search")#sets the main button for opening the
    Search GUI and function
37.         self.print_push_button = QPushButton("Print")#sets the main button for opening the Print
    GUI and function
38.
39.         self.tab_bar = QTabWidget() #creates the widget to display the tables in a tabular layout
40.
41.         self.tabs = {}#creates a dictionary for the table names/tab names
42.         self.tabNames = ['Members','Payments','Regime','Exercise']#The 4 tab/table names
43.         for count in range(4):
44.             self.tabs["{0}".format(self.tabNames[count])] = BrowseDataWidget()
45.
    self.tab_bar.addTab(self.tabs["{0}".format(self.tabNames[count])],"{0}".format(self.tabNames
    [count]))
46.         #a for loop that creates a new tab and adds a "BrowseDataWidget" into each of them
47.
48.         self.labels = {"Members":["MemberID","Name","Address","Telephone
    Number","Membership Type","Induction Date","Join Date","How Paid", "Amount",
    "RegistrationFee","Registration Date","PaymentType","Comments"],
49.                        "Payments":["MemberID","Payment Date","How Much","Paid"],
50.                        "Regime":["MemberID","ExerciseID","Specific Description", "Start Date", "End
    Date"],
```

```
                "Exercise":["ExerciseID","Name","Description"]}
        #labels for the table headers that are referenced later in the program

        self.toolBar = QMenuBar()#creates a menu bar
        self.file_menu = self.toolBar.addMenu("File")#adds File option to the tool bar
        self.help_menu = self.toolBar.addMenu("Help")#adds Help option to the tool bar
        self.about = self.help_menu.addAction("About Gym Database Management System
9001")#adds options into the Help menu
        self.open_shortcut = self.file_menu.addAction("Open")#adds Open shortcut to the File
menu
        self.add_shortcut = self.file_menu.addAction("Add")#adds Add shortcut to the File menu
        self.edit_shortcut = self.file_menu.addAction("Edit")#adds Edit shortcut to the File menu
        self.delete_shortcut = self.file_menu.addAction("Delete")#adds Delete shortcut to the File
menu
        self.search_shortcut = self.file_menu.addAction("Search")#adds Search shortcut to the File
menu
        self.print_shortcut = self.file_menu.addAction("Print")#adds Print shortcut to the File menu


        #layout
        self.layout1 = QHBoxLayout()
        self.layout2 = QHBoxLayout()
        self.layout3 = QVBoxLayout()
        self.layout1.addWidget(self.open_push_button)
        self.layout1.addWidget(self.add_push_button)
        self.layout1.addWidget(self.edit_push_button)

        self.layout2.addWidget(self.delete_push_button)
        self.layout2.addWidget(self.search_push_button)
        self.layout2.addWidget(self.print_push_button)

        self.layout3.addWidget(self.tab_bar)
        self.layout3.addLayout(self.layout1)
```

```
80.         self.layout3.addLayout(self.layout2)
81.
82.         self.mainWidget = QWidget()
83.         self.setMenuWidget(self.toolBar)
84.         self.mainWidget.setLayout(self.layout3)
85.         self.setCentralWidget(self.mainWidget)
86.
87.         #connections
88.         #connections linking the main pushbuttons to their respective functions
89.         self.open_push_button.clicked.connect(self.open_file_menu)
90.         self.delete_push_button.clicked.connect(self.delete)
91.         self.print_push_button.clicked.connect(self.print_stuff)
92.         self.search_push_button.clicked.connect(self.search)
93.         self.edit_push_button.clicked.connect(self.edit)
94.         self.add_push_button.clicked.connect(self.add)
95.         self.about.triggered.connect(self.about_the_program)
96.         self.open_shortcut.triggered.connect(self.open_file_menu)
97.         self.add_shortcut.triggered.connect(self.add)
98.         self.edit_shortcut.triggered.connect(self.edit)
99.         self.delete_shortcut.triggered.connect(self.delete)
100.             self.search_shortcut.triggered.connect(self.search)
101.             self.print_shortcut.triggered.connect(self.print_stuff)
102.             #Keyboard Shortcuts for accessing the 6 main Functions
103.             self.connect(QShortcut(QKeySequence("Ctrl+o"), self), SIGNAL('activated()'),
    self.open_file_menu)
104.             self.connect(QShortcut(QKeySequence("Ctrl+a"), self), SIGNAL('activated()'), self.add)
105.             self.connect(QShortcut(QKeySequence("Ctrl+e"), self), SIGNAL('activated()'), self.edit)
106.             self.connect(QShortcut(QKeySequence("Ctrl+d"), self), SIGNAL('activated()'),
    self.delete)
107.             self.connect(QShortcut(QKeySequence("Ctrl+s"), self), SIGNAL('activated()'),
    self.search)
108.             self.connect(QShortcut(QKeySequence("Ctrl+p"), self), SIGNAL('activated()'),
    self.print_stuff)
```

```
109.            self.connect(QShortcut(QKeySequence("Ctrl+h"), self), SIGNAL('activated()'),
    self.about_the_program)
110.
111.
112.
113.        def open_file_menu(self):
114.            self.file = QFileDialog.getOpenFileName(caption="Open Database",filter = "Database
    file (*.db *.dat)")#opens the windows folder tree allowing the user to select the database they
    want to open. File type restricted to .db or .dat files
115.            try: #Restricts the user to only opening correct database or a version of it
116.                for item in self.tabNames:
117.                    self.tabs["{0}".format(item)].UpdateTable(self.file)#loads selected database into
    tabs
118.                    self.tabs["{0}".format(item)].PopulateTable(item, self.labels[item])#populates
    the tabs with relevent information
119.            except NameError:
120.                return
121.            except sqlite3.OperationalError:
122.                return
123.
124.
125.        def delete(self):
126.            self.password("niel")#sets delete password to "niel" and opens the password function
127.            delete_dialog = DeleteDialog(self.file)#opens delete dialog
128.            delete_dialog.exec_()
129.
130.        def print_stuff(self):
131.            print_dialog = PrintDialog(self.file)#opens print dialog
132.            print_dialog.exec_()
133.
134.        def search(self):
135.            search_dialog = SearchDialog(self.file)#opens search dialog
136.            search_dialog.exec_()
```

```
137.
138.     def edit(self):
139.         edit_dialog = EditDialog(self.file)#opens edit dialog
140.         edit_dialog.exec_()
141.
142.     def add(self):
143.         add_dialog = AddDialog(self.file)#opens add dialog
144.         add_dialog.exec_()
145.
146.     def about_the_program(self):
147.         about_dialog = AboutDialog()#opens about dialog
148.         about_dialog.exec_()
149.
150.     def password(self,currentPass):
151.         passWord = ""#sets entered password to the wrong password
152.         while passWord != currentPass:#while the correct password hasn't been entered
153.             password_dialog = PasswordDialog()#opens password dialog
154.             password_dialog.exec_()
155.             passWord = password_dialog.close_method()
156.
157.
158. def main():
159.     gym_program = QApplication(sys.argv)#creates application
160.     gym_window = AppWindow()#creates Main Window
161.     gym_window.resize(700,600)#Locks inital window size
162.     password_dialog = gym_window.password("sweatGym9001_niel")
163.     gym_window.show()
164.     gym_window.raise_()
165.     gym_program.exec_()
166.
167.
168. if __name__ == "__main__":
169.     main()
```

AddEditMemberTableWIdget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditMemberTableWidget(QWidget):
6.      """This class creates the widget for the members table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(13):
16.             self.lineEdits["self.enter_text {0}".format(count)] = QLineEdit("")#creates 13
    QLineEdits, 1 for each column in the table
17.         self.memberID_Label = QLabel("MemberID")
18.         self.name_Label = QLabel("Name")
19.         self.address_Label = QLabel("Address")
20.         self.telephone_number_Label = QLabel("Telephone Number")
21.         self.membership_type_Label = QLabel("Membership Type")
22.         self.induction_date_Label = QLabel("Induction Date")
23.         self.join_date_Label = QLabel("Join Date")
24.         self.how_paid_Label = QLabel("How Paid")
25.         self.amount_Label = QLabel("Amount")
26.         self.registration_fee_Label = QLabel("Registration Fee")
27.         self.registration_date_Label = QLabel("Registration Date")
28.         self.payment_type_Label = QLabel("Payment Type")
29.         self.comments_Label = QLabel("Comments")
```

```
30.
31.         #create member layout
32.
33.         self.mainMemberLayout = QVBoxLayout()
34.         self.memberContentLayout = QHBoxLayout()
35.         self.memberLabelLayout = QVBoxLayout()
36.         self.memberInputLayout = QVBoxLayout()
37.
38.         self.memberLabelLayout.addWidget(self.memberID_Label)
39.         self.memberLabelLayout.addWidget(self.name_Label)
40.         self.memberLabelLayout.addWidget(self.address_Label)
41.         self.memberLabelLayout.addWidget(self.telephone_number_Label)
42.         self.memberLabelLayout.addWidget(self.membership_type_Label)
43.         self.memberLabelLayout.addWidget(self.induction_date_Label)
44.         self.memberLabelLayout.addWidget(self.join_date_Label)
45.         self.memberLabelLayout.addWidget(self.how_paid_Label)
46.         self.memberLabelLayout.addWidget(self.amount_Label)
47.         self.memberLabelLayout.addWidget(self.registration_fee_Label)
48.         self.memberLabelLayout.addWidget(self.registration_date_Label)
49.         self.memberLabelLayout.addWidget(self.payment_type_Label)
50.         self.memberLabelLayout.addWidget(self.comments_Label)
51.         for count in range(13):
52.             self.memberInputLayout.addWidget(self.lineEdits["self.enter_text
    {0}".format(count)])#adds each QLineEdit to the layout
53.
54.         self.memberContentLayout.addLayout(self.memberLabelLayout)
55.         self.memberContentLayout.addLayout(self.memberInputLayout)
56.         self.mainMemberLayout.addLayout(self.memberContentLayout)
57.
58.         self.setLayout(self.mainMemberLayout)
59.
60.     def addMemberItems(self,database):
61.             #method for adding items to the member table
```

```
62.         addMember(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
    1"].text(),self.lineEdits["self.enter_text 2"].text(),self.lineEdits["self.enter_text
    3"].text(),self.lineEdits["self.enter_text 4"].text(),self.lineEdits["self.enter_text
    5"].text(),self.lineEdits["self.enter_text 6"].text(),self.lineEdits["self.enter_text
    7"].text(),self.lineEdits["self.enter_text 8"].text(),self.lineEdits["self.enter_text
    9"].text(),self.lineEdits["self.enter_text 10"].text(),self.lineEdits["self.enter_text
    11"].text(),self.lineEdits["self.enter_text 12"].text())
63.
64.     def editMemberItems(self,database):
65.             #method for editting items in the member table
66.         columns
    =["Name","Address","TelephoneNumber","MembershipType","InductionDate","JoinDate","HowP
    aid","Amount","RegistrationFee","RegistrationDate","PaymentType","Comments"]
67.         items = ""
68.         for count in range(1,12):
69.             if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
70.                 if count == 1 or 2 or 3 or 4 or 8 or 11 or 12:
71.                     items +=(columns[count-1] + "='" + self.lineEdits["self.enter_text
    {0}".format(count)].text() + "' ,")
72.                 else:
73.                     items +=(columns[count-1] + "=" + self.lineEdits["self.enter_text
    {0}".format(count)].text() + " ,")
74.         items = items[:-1]
75.
76.         editMember(database,items,self.lineEdits["self.enter_text 0"].text())
```

AddEditPaymentTableWidget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditPaymentTableWidget(QWidget):
6.      """This class creates the widget for the payment table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(4):
16.             self.lineEdits["self.enter_text {0}".format(count)] = QLineEdit()#creates 4 QLineEdits, 1
    for each column in the payments table
17.
18.         self.memberID_Label = QLabel("MemberID")
19.         self.payment_date_Label = QLabel("Payment Date")
20.         self.how_much_Label = QLabel("How Much")
21.         self.paid_label = QLabel("Paid")
22.
23.         # create payment layout
24.
25.         self.paymentLayout = QVBoxLayout()
26.         self.paymentContentLayout = QHBoxLayout()
27.         self.paymentLabelLayout = QVBoxLayout()
28.         self.paymentInputLayout = QVBoxLayout()
29.
30.         self.paymentLabelLayout.addWidget(self.memberID_Label)
```

```
31.         self.paymentLabelLayout.addWidget(self.payment_date_Label)
32.         self.paymentLabelLayout.addWidget(self.how_much_Label)
33.         self.paymentLabelLayout.addWidget(self.paid_label)
34.         for count in range(4):
35.             self.paymentInputLayout.addWidget(self.lineEdits["self.enter_text
    {0}".format(count)])#adds all the QLineEdits to the layout
36.
37.         self.paymentContentLayout.addLayout(self.paymentLabelLayout)
38.         self.paymentContentLayout.addLayout(self.paymentInputLayout)
39.         self.paymentLayout.addLayout(self.paymentContentLayout)
40.
41.         self.setLayout(self.paymentLayout)
42.
43.     def addPaymentItems(self, database):
44.             #method for adding items to the payment table
45.         addPayment(database,self.lineEdits["self.enter_text
    0"].text(),self.lineEdits["self.enter_text 1"].text(),self.lineEdits["self.enter_text
    2"].text(),self.lineEdits["self.enter_text 3"].text())
46.
47.     def editPaymentItems(self,database):
48.             #method for editing items in the payment table
49.         columns = ["HowMuch","Paid"]
50.         items = ""
51.         for count in range(2,4):
52.             print(count)
53.             if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
54.                 items +=(columns[count-2] + "=" + self.lineEdits["self.enter_text
    {0}".format(count)].text() + " ,")
55.         items = items[:-1]
56.
57.         editPayment(database,items,self.lineEdits["self.enter_text
    0"].text(),self.lineEdits["self.enter_text 1"].text())
```

AddEditRegimeTableWIdget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditRegimeTableWidget(QWidget):
6.      """This class creates the widget for the regime table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(5):
16.             self.lineEdits["self.enter_text {0}".format(count)] = QLineEdit()#creates 5 QLineEdits, 1
    for each column in the regime database
17.
18.         self.memberID_Label = QLabel("MemberID")
19.         self.exerciseID_Label = QLabel("Exercise ID")
20.         self.start_date_Label = QLabel("Start Date")
21.         self.end_date_Label = QLabel("End Date")
22.         self.description_Label = QLabel("Description")
23.
24.         # create regime layout
25.
26.         self.regimeLayout = QVBoxLayout()
27.         self.regimeContentLayout = QHBoxLayout()
28.         self.regimeLabelLayout = QVBoxLayout()
29.         self.regimeInputLayout = QVBoxLayout()
```

```
30.
31.         self.regimeLabelLayout.addWidget(self.memberID_Label)
32.         self.regimeLabelLayout.addWidget(self.exerciseID_Label)
33.         self.regimeLabelLayout.addWidget(self.start_date_Label)
34.         self.regimeLabelLayout.addWidget(self.end_date_Label)
35.         self.regimeLabelLayout.addWidget(self.description_Label)
36.         for count in range(5):
37.             self.regimeInputLayout.addWidget(self.lineEdits["self.enter_text
    {0}".format(count)])#adds each QLineEdit to the layout
38.
39.         self.regimeContentLayout.addLayout(self.regimeLabelLayout)
40.         self.regimeContentLayout.addLayout(self.regimeInputLayout)
41.         self.regimeLayout.addLayout(self.regimeContentLayout)
42.
43.         self.setLayout(self.regimeLayout)
44.
45.
46.     def addRegimeItems(self,database):
47.             #method for adding items to the regime table
48.         addRegime(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
    1"].text(),self.lineEdits["self.enter_text 2"].text(),self.lineEdits["self.enter_text
    3"].text(),self.lineEdits["self.enter_text 3"].text())
49.
50.     def editRegimeItems(self,database):
51.             #method for editing items in the regime table
52.         columns = ["StartDate","EndDate","SpecificDescription"]
53.         items = ""
54.         for count in range(2,5):
55.             if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
56.                 if count == 4:
57.                     items +=(columns[count-2] + "='" + self.lineEdits["self.enter_text
    {0}".format(count)].text() + "' ,")
58.                 else:
```

```
59.                 items +=(columns[count-2] + "=" + self.lineEdits["self.enter_text
{0}".format(count)].text() + " ,")
60.         items = items[:-1]
61.
62.         editRegime(database,items,self.lineEdits["self.enter_text
0"].text(),self.lineEdits["self.enter_text 1"].text())
```

AddEditExerciseTableWidget

```
1.  from PyQt4.QtGui import *
2.  from gym_add_function import *
3.  from gym_edit_function import *
4.
5.  class AddEditExerciseTableWidget(QWidget):
6.      """This class creates the widget for the exercise table in the add and edit dialog boxes"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.
13.         self.lineEdits = {}
14.
15.         for count in range(3):
16.             self.lineEdits["self.enter_text {0}".format(count)] = QLineEdit()#creates 3 QLineEdits, 1
    for each column in the table
17.
18.         self.exerciseID_Label = QLabel("Exercise ID")
19.         self.name_Label = QLabel("Name")
20.         self.description_Label = QLabel("Description")
21.
22.
23.         # create exercise layout
24.         self.exerciseLayout = QVBoxLayout()
25.         self.exerciseContentLayout = QHBoxLayout()
26.         self.exerciseLabelLayout = QVBoxLayout()
27.         self.exerciseInputLayout = QVBoxLayout()
28.
29.         self.exerciseLabelLayout.addWidget(self.exerciseID_Label)
30.         self.exerciseLabelLayout.addWidget(self.name_Label)
```

```
31.         self.exerciseLabelLayout.addWidget(self.description_Label)
32.
33.         for count in range(3):
34.             self.exerciseInputLayout.addWidget(self.lineEdits["self.enter_text
    {0}".format(count)])#adds each QLineEdit to the layout
35.
36.         self.exerciseContentLayout.addLayout(self.exerciseLabelLayout)
37.         self.exerciseContentLayout.addLayout(self.exerciseInputLayout)
38.         self.exerciseLayout.addLayout(self.exerciseContentLayout)
39.
40.         self.setLayout(self.exerciseLayout)
41.
42.     def addExerciseItems(self, database):
43.             #method for adding items to the exercise table
44.         addExercise(database,self.lineEdits["self.enter_text 0"].text(),self.lineEdits["self.enter_text
    1"].text(),self.lineEdits["self.enter_text 2"].text())
45.
46.
47.     def editExerciseItems(self,database):
48.             #method for editing items in the exercise table
49.         columns = ["Name","Description"]
50.         items = ""
51.         for count in range(1,3):
52.             if self.lineEdits["self.enter_text {0}".format(count)].text() != "":
53.                 items +=(columns[count-1] + "='" + self.lineEdits["self.enter_text
    {0}".format(count)].text() + "' ,")
54.         items = items[:-1]
55.
56.         editExercise(database,items,self.lineEdits["self.enter_text 0"].text())
```

BrowseDatabaseWidget

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtSql import *
3.  from sqlite3 import *
4.
5.  from gym_database_class import *
6.
7.  class BrowseDataWidget(QWidget):
8.      """A widget for displaying Database data"""
9.
10.     def __init__(self):
11.
12.         self.loadDataBase = None
13.         super().__init__()
14.         self.layout = QVBoxLayout()
15.
16.         self.table_view = QTableView()
17.
18.         self.layout.addWidget(self.table_view)
19.
20.         self.setLayout(self.layout)
21.
22.         self.database = None
23.
24.
25.     def PopulateTable(self,item,labels):
26.             #method for adding all the information from the database to the table view based on
    the currently opened database
27.             self.database = Database(self.loadDataBase)
28.             self.database.loadDatabase()
29.             data = self.database.getAllData(item)
30.             model = QStandardItemModel()
```

```
            model.setHorizontalHeaderLabels(labels)
            row = 0
            for item in data:
                for column in range(len(item)):
                    if item[column] == "None":
                        print(item[column])
                        item = (" ")
                    else:
                        StandardItem = QStandardItem("{}".format(item[column]))
                        model.setItem(row, column, StandardItem)
                row += 1
            self.table_view.setModel(model)



    def UpdateTable(self,newDataBase):
            #method for loading a new database when the user opens a new database or reopens a database
        self.loadDataBase = newDataBase
```

Print Function

```
1.  import sqlite3
2.
3.  def getMemberInfo(database,memberID):
4.      #gets the items from the database for a member info printout
5.
6.      con = sqlite3.connect(database)
7.
8.      with con:
9.
10.         cur = con.cursor()
11.         cur.execute("SELECT * FROM MEMBERS WHERE MEMBERID = "+memberID)
12.         result = cur.fetchall()
13.         return result
14.
15. def getInvoice(database,memberID):
16.     #gets the items from the database for a print out of an invoice
17.
18.     con = sqlite3.connect(database)
19.
20.     with con:
21.
22.         cur = con.cursor()
23.         cur.execute("SELECT * FROM PAYMENTS WHERE MEMBERID = "+memberID)
24.         result = cur.fetchall()
25.         cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID = "+memberID)
26.         name = cur.fetchall()
27.
28.         return result,name
29.
30. def getRegime(database,memberID):
31.     #gets the items from the database for an invoice printout
32.
```

```
33.     con = sqlite3.connect(database)
34.
35.     with con:
36.
37.         cur = con.cursor()
38.         cur.execute("SELECT * FROM REGIME WHERE MEMBERID = "+memberID)
39.         result = cur.fetchall()
40.         cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID = "+memberID)
41.         name = cur.fetchall()
42.
43.         return result,name
```

Search Function

```
1.  import sqlite3
2.
3.  def searchQuery(database,table,constraint):
4.      #function that searches the correct table in the database with the correct constraints
5.
6.      con = sqlite3.connect(database)
7.
8.      with con:
9.
10.         cur = con.cursor()
11.         if table == "MEMBERS":
12.             cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR Name
    Like "+constraint+" OR Address Like "+constraint+" OR TelephoneNumber Like "+constraint+"
    OR MembershipType Like "+constraint+" OR InductionDate Like "+constraint+" OR JoinDate Like
    "+constraint+" OR HowPaid Like "+constraint+" OR Amount Like "+constraint+" OR
    RegistrationFee Like "+constraint+" OR RegistrationDate Like "+constraint+" OR PaymentType
    Like "+constraint+" OR Comments Like "+constraint)
13.             result = cur.fetchall()
14.             return result
15.         if table == "PAYMENTS":
16.             cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR
    PaymentDate Like "+constraint+" OR HowMuch Like "+constraint+" OR Paid Like "+constraint)
17.             result = cur.fetchall()
18.             return result
19.         if table == "REGIME":
20.             cur.execute("SELECT * FROM "+table+" WHERE MEMBERID Like "+constraint+" OR
    EXERCISEID Like "+constraint+" OR SpecificDescription Like "+constraint+" OR StartDate Like
    "+constraint+" OR EndDate Like "+constraint)
21.             result = cur.fetchall()
22.             return result
23.         if table == "EXERCISE":
```

```
24.             cur.execute("SELECT * FROM "+table+" WHERE EXERCISEID Like "+constraint+" OR
    Name Like "+constraint+" OR Description Like "+constraint)
25.             result = cur.fetchall()
26.             return result
```

Add Function

```
1.  import sqlite3
2.
3.  def addMember(database, memberID, name, address, telNumber, membershipType,
    inductionDate, joinDate, howPaid, amount, registrationDate,registrationFee, paymentType,
    comments):
4.      #function for adding items to the member table
5.      con = sqlite3.connect(database)
6.
7.      with con:
8.
9.          sql = "INSERT INTO Members (MemberID, Name, Address, TelephoneNumber,
    MembershipType, InductionDate, JoinDate, HowPaid, Amount, RegistrationDate, RegistrationFee,
    PaymentType, Comments) VALUES
    ('"+memberID+"','"+name+"','"+address+"','"+telNumber+"','"+membershipType+"','"+inducti
    onDate+"','"+joinDate+"','"+howPaid+"','"+amount+"','"+registrationDate+"','"+registrationFe
    e+"','"+paymentType+"','"+comments+"')"
10.         print(sql)
11.         cur = con.cursor()
12.         cur.execute(sql)
13.
14.
15. def addPayment(database,memberID,paymentDate,howMuch,paid):
16.     #function for adding items to the payment table
17.     con = sqlite3.connect(database)
18.
19.     with con:
20.
21.         cur = con.cursor()
22.         cur.execute("INSERT INTO Payments (MemberID, PaymentDate, HowMuch, Paid) VALUES
    ("+memberID+",'"+paymentDate+"',"+howMuch+","+paid+")")
23.
24. def addRegime(database,memberID,exerciseID,specificDescription,startDate,endDate):
```

```
25.     #function for adding items to the regime table
26.     con = sqlite3.connect(database)
27.
28.     with con:
29.
30.         cur = con.cursor()
31.         cur.execute("INSERT INTO Regime(MemberID, ExerciseID, SpecificDescription, StartDate,
    EndDate) VALUES
    ("+memberID+","+exerciseID+",'"+specificDescription+"','"+startDate+"','"+endDate+"')")
32.
33. def addExercise(database,exerciseID,name,description):
34.     #function for adding items to the exercise table
35.     con = sqlite3.connect(database)
36.
37.     with con:
38.
39.         cur = con.cursor()
40.         cur.execute("INSERT INTO Exercise(ExerciseID, Name, Description) VALUES
    ("+exerciseID+",'"+name+"','"+description+"')")
```

Edit Function

```
1.  import sqlite3
2.
3.  def editMember(database,items,memberID):
4.      #function for editing items in the member table
5.      con = sqlite3.connect(database)
6.
7.      with con:
8.
9.          sql = "UPDATE Members SET "+items+" WHERE MemberID = "+memberID
10.         cur = con.cursor()
11.         cur.execute(sql)
12.
13. def editPayment(database,items,memberID,paymentDate):
14.     #function for editing items in the payment table
15.     con = sqlite3.connect(database)
16.
17.     with con:
18.         sql = "UPDATE Payments SET "+items+" WHERE MemberID="+memberID+" AND
    PaymentDate = "+paymentDate
19.         cur = con.cursor()
20.         cur.execute(sql)
21.
22. def editRegime(database,items,memberID,exerciseID):
23.     #function for editing items in the regimetable
24.     con = sqlite3.connect(database)
25.
26.     with con:
27.         sql = "UPDATE Regime SET "+items+" WHERE MemberID = "+memberID+" AND ExerciseID
    = "+exerciseID
28.         cur = con.cursor()
29.         cur.execute(sql)
30.
```

```
31. def editExercise(database,items,exerciseID):
32.     #function for editing items in the exercise table
33.     con = sqlite3.connect(database)
34.
35.     with con:
36.         sql = "UPDATE Exercise SET "+items+"WHERE EXERCISEID = "+exerciseID
37.         cur = con.cursor()
38.         cur.execute(sql)
```

Delete Function

```
1.  import sqlite3
2.
3.  def deleteQueryPrimaryKey(database,table,item,constraint):
4.      #method for deleting items from the Members or Exercise Tables as they have a primary key
5.
6.      con = sqlite3.connect(database)
7.
8.      with con:
9.
10.         cur = con.cursor()
11.         cur.execute("DELETE FROM "+table+" WHERE "+item+" = "+constraint)
12.
13. def deleteQueryCompositeKey(database,table,item1,item2,constraint1,constraint2):
14.     #method for deleting items from the payments or regimes tables as they have composite keys
15.     con = sqlite3.connect(database)
16.
17.     with con:
18.
19.         cur = con.cursor()
20.         sql = "DELETE FROM "+table+" WHERE "+item1+" = "+constraint1+" AND "+item2+" =
    "+constraint2
21.         print(sql)
22.         cur.execute(sql)
23.
24. def deleteAll(database,table):
25.         #delete all items from any table
26.     con = sqlite3.connect(database)
27.
28.     with con:
29.
30.         cur = con.cursor()
31.         cur.execute("DELETE FROM "+table)
```

```
32.
33.
34. def getItems(database,table):
35.     #function to retrieve all the data from the tables
36.
37.     con = sqlite3.connect(database)
38.
39.     with con:
40.
41.         cur = con.cursor()
42.         cur.execute("SELECT * FROM "+table)
43.
44.         results = cur.fetchall()
45.         items = []
46.
47.         for count in range(len(results)):
48.             column = results[count]
49.             if table == "MEMBERS":
50.                 items.append(str(column[0])+". "+str(column[1]))
51.             if table == "PAYMENTS":
52.                 cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID="+str(column[0]))
53.                 memberName = cur.fetchall()
54.                 for part in memberName:
55.                     items.append(str(column[0])+". "+str(part[0])+" - "+str(column[1]))
56.             if table == "REGIME":
57.                 cur.execute("SELECT Name FROM MEMBERS WHERE MEMBERID="+str(column[0]))
58.                 memberName = cur.fetchall()
59.                 cur.execute("SELECT Name FROM EXERCISE WHERE EXERCISEID="+str(column[1]))
60.                 exerciseName = cur.fetchall()
61.                 name = ""
62.                 exercise = ""
63.                 for part in memberName:
64.                     name = str(part[0])
```

```
            for part in exerciseName:
                exercise = str(part[0])
            items.append(str(column[0])+". "+name+" - "+str(column[1])+". "+exercise)
        if table == "EXERCISE":
            items.append(str(column[0])+". "+str(column[1]))

    return items
```

Database Class

```
1.  import sqlite3
2.
3.  class Database:
4.      def __init__(self, db_name):
5.          self.db_name = db_name
6.
7.      def loadDatabase(self):
8.          #method that connects to the right database based on the file that was opened by the user
9.          with sqlite3.connect(self.db_name) as db:
10.             cursor = db.cursor()
11.
12.     def getAllData(self,table):
13.         #method for retrieving all the data from a specific table in the database with a Select Query
14.         with sqlite3.connect(self.db_name) as db:
15.             allData = []
16.             cursor = db.cursor()
17.             query = "Select * From " + str(table)
18.             cursor.execute(query)
19.             data = cursor.fetchall()
20.             for item in data:
21.                 allData.append(item)
22.             return allData
```

AddEdit Dialog

```
1.  from PyQt4.QtGui import *
2.  from add_edit_member_table_widget_class import *
3.  from add_edit_payment_table_widget_class import *
4.  from add_edit_regime_table_widget_class import *
5.  from add_edit_exercise_table_widget_class import *
6.  from gym_add_function import *
7.
8.  class AddEditDialog(QDialog):
9.      """This class creates the dialog window for Adding items to the database and answering
    them"""
10.
11.     def __init__(self, database):
12.         super().__init__()
13.
14.         self.table_combo_box = QComboBox()
15.         self.table_combo_box.addItem("Members")
16.         self.table_combo_box.addItem("Payments")
17.         self.table_combo_box.addItem("Regimes")
18.         self.table_combo_box.addItem("Exercises")
19.         self.add_edit_button = QPushButton()
20.         self.database = database
21.
22.         self.stackedLayout = QStackedLayout()
23.         self.mainLayout = QVBoxLayout()
24.         self.topLayout = QHBoxLayout()
25.
26.         self.setWindowTitle(" ")
27.         self.setWindowIcon(QIcon("WindowIcon.png"))
28.
29.         self.memberLayoutWidget = AddEditMemberTableWidget()
30.         self.paymentLayoutWidget = AddEditPaymentTableWidget()
31.         self.regimeLayoutWidget = AddEditRegimeTableWidget()
```

```
32.         self.exerciseLayoutWidget = AddEditExerciseTableWidget()
33.         self.stackedLayout.addWidget(self.memberLayoutWidget)
34.         self.stackedLayout.addWidget(self.paymentLayoutWidget)
35.         self.stackedLayout.addWidget(self.regimeLayoutWidget)
36.         self.stackedLayout.addWidget(self.exerciseLayoutWidget)
37.
38.         self.topLayout.addWidget(self.table_combo_box)
39.         self.topLayout.addWidget(self.add_edit_button)
40.         self.mainLayout.addLayout(self.topLayout)
41.         self.mainLayout.addLayout(self.stackedLayout)
42.
43.         self.setLayout(self.mainLayout)
44.
45.         #connections
46.         self.table_combo_box.currentIndexChanged.connect(self.updateTable)#when the
    comboboxes currently selected table is changed the updateTable method is run
47.
48.
49.     def updateTable(self,index):
50.             #method for chaning the widget based on the table selected in the combobox
51.         self.stackedLayout.setCurrentIndex(index)
```

Add Dialog

```
1.  from PyQt4.QtGui import *
2.  from gym_add_edit_dialog_class import *
3.
4.  class AddDialog(AddEditDialog):
5.      """This class creates the dialog window for Adding items to the database"""
6.
7.      def __init__(self,database):
8.          super().__init__(database)
9.
10.         self.add_edit_button.setText("Add")
11.         self.setWindowTitle("Add")
12.
13.         #connections
14.         self.add_edit_button.clicked.connect(self.addItems)
15.
16.     def addItems(self):
17.             #method for using the correct methods to add items to a table, with the
    table changing based on which widget is selected from the stacked layout
18.         if self.stackedLayout.currentIndex() == 0:
19.             self.memberLayoutWidget.addMemberItems(self.database)
20.         if self.stackedLayout.currentIndex() == 1:
21.             self.paymentLayoutWidget.addPaymentItems(self.database)
22.         if self.stackedLayout.currentIndex() == 2:
23.             self.regimeLayoutWidget.addRegimeItems(self.database)
24.         if self.stackedLayout.currentIndex() == 3:
25.             self.exerciseLayoutWidget.addExerciseItems(self.database)
26.         self.close()
```

Edit Dialog

```
1.  from PyQt4.QtGui import *
2.  from gym_add_edit_dialog_class import *
3.
4.  class EditDialog(AddEditDialog):
5.      """This class creates the dialog window for editing items in the database"""
6.
7.      def __init__(self,database):
8.          super().__init__(database)
9.
10.         self.add_edit_button.setText("Edit")
11.         self.setWindowTitle("Edit")
12.
13.         #connections
14.         self.add_edit_button.clicked.connect(self.editItems)
15.
16.     def editItems(self):
17.             #method for using the correct methods to edit items in a table, with the table changing
    based on which widget is selected from the stacked layout
18.         if self.stackedLayout.currentIndex() == 0:
19.             self.memberLayoutWidget.editMemberItems(self.database)
20.         if self.stackedLayout.currentIndex() == 1:
21.             self.paymentLayoutWidget.editPaymentItems(self.database)
22.         if self.stackedLayout.currentIndex() == 2:
23.             self.regimeLayoutWidget.editRegimeItems(self.database)
24.         if self.stackedLayout.currentIndex() == 3:
25.             self.exerciseLayoutWidget.editExerciseItems(self.database)
26.         self.close()
27.
```

Delete Dialog

```
1.  from PyQt4.QtGui import *
2.  from gym_delete_function import *
3.
4.  class DeleteDialog(QDialog):
5.      """This class creates the dialog window for deleting items from the database"""
6.
7.      def __init__(self,database):
8.          super().__init__()
9.
10.         self.database = database
11.
12.         #create widgets
13.         self.table_select_combo_box = QComboBox()
14.         self.item_select_combo_box = QComboBox()
15.         self.delete_push_button = QPushButton("Delete")
16.         self.delete_all_push_button = QPushButton("Delete All Items")
17.
18.         self.table_select_combo_box.addItem("Select Table")
19.         self.table_select_combo_box.addItem("Members")
20.         self.table_select_combo_box.addItem("Payments")
21.         self.table_select_combo_box.addItem("Regimes")
22.         self.table_select_combo_box.addItem("Exercises")
23.
24.         self.item_select_combo_box.addItem("Select Item")
25.
26.         #create layout
27.         self.layout = QVBoxLayout()
28.
29.         #add widgets to layout
30.         self.layout.addWidget(self.table_select_combo_box)
31.         self.layout.addWidget(self.item_select_combo_box)
32.         self.layout.addWidget(self.delete_push_button)
```

```
33.         self.layout.addWidget(self.delete_all_push_button)
34.
35.         #set the window layout
36.         self.setLayout(self.layout)
37.         self.setWindowTitle("Delete")
38.         self.setWindowIcon(QIcon("WindowIcon.png"))
39.
40.         #connections
41.         self.table_select_combo_box.currentIndexChanged.connect(self.itemComboBoxPopulate)
42.         self.delete_push_button.clicked.connect(self.deleteItems)
43.         self.delete_all_push_button.clicked.connect(self.deleteAllItems)
44.
45.     def itemComboBoxPopulate(self):
46.             #method for populating the item select combobox with the correct items and formating
    from the correct tables
47.         if self.table_select_combo_box.currentIndex() == 0:
48.             self.item_select_combo_box.clear()
49.             self.item_select_combo_box.addItem("Select Item")
50.         if self.table_select_combo_box.currentIndex() == 1:
51.             items = getItems(self.database, "MEMBERS")
52.             self.item_select_combo_box.clear()
53.             for count in range(len(items)):
54.                 self.item_select_combo_box.addItem(items[count])
55.         if self.table_select_combo_box.currentIndex() == 2:
56.             items = getItems(self.database, "PAYMENTS")
57.             self.item_select_combo_box.clear()
58.             for count in range(len(items)):
59.                 self.item_select_combo_box.addItem(items[count])
60.         if self.table_select_combo_box.currentIndex() == 3:
61.             items = getItems(self.database, "REGIME")
62.             self.item_select_combo_box.clear()
63.             for count in range(len(items)):
64.                 self.item_select_combo_box.addItem(items[count])
```

```
65.         if self.table_select_combo_box.currentIndex() == 4:
66.             items = getItems(self.database, "EXERCISE")
67.             self.item_select_combo_box.clear()
68.             for count in range(len(items)):
69.                 self.item_select_combo_box.addItem(items[count])
70.
71.     def deleteItems(self):
72.                 #method for deleting the correct items from the correct table
73.         if self.table_select_combo_box.currentIndex() == 0:
74.             return
75.         if self.table_select_combo_box.currentIndex() == 1:
76.
deleteQueryPrimaryKey(self.database,"MEMBERS","MEMBERID",str(self.item_select_combo_box.
currentText())[0])
77.         if self.table_select_combo_box.currentIndex() == 2:
78.             sep = " - "
79.
deleteQueryCompositeKey(self.database,"PAYMENTS","MEMBERID","PAYMENTDATE",str(self.ite
m_select_combo_box.currentText())[0],str(self.item_select_combo_box.currentText()).split(sep
,1)[1])
80.         if self.table_select_combo_box.currentIndex() == 3:
81.             sep = " - "
82.
deleteQueryCompositeKey(self.database,"REGIME","MEMBERID","EXERCISEID",str(self.item_sel
ect_combo_box.currentText())[0],str(self.item_select_combo_box.currentText()).split(sep,1)[1]
[0])
83.         if self.table_select_combo_box.currentIndex() == 4:
84.
deleteQueryPrimaryKey(self.database,"EXERCISE","EXERCISEID",str(self.item_select_combo_bo
x.currentText())[0])
85.
86.     def deleteAllItems(self):
87.                 #method for deleting every item in a certain table
```

```
88.         if self.table_select_combo_box.currentIndex() == 0:
89.             return
90.         if self.table_select_combo_box.currentIndex() == 1:
91.             deleteAll(self.database,"MEMBERS")
92.         if self.table_select_combo_box.currentIndex() == 2:
93.             deleteAll(self.database,"PAYMENTS")
94.         if self.table_select_combo_box.currentIndex() == 3:
95.             deleteAll(self.database,"REGIME")
96.         if self.table_select_combo_box.currentIndex() == 4:
97.             deleteAll(self.database,"EXERCISE")
```

Search Dialog

```
1.  from PyQt4.QtGui import *
2.  from gym_search_results_dialog_class import *
3.  from gym_search_function import *
4.
5.  class SearchDialog(QDialog):
6.      """This class creates the dialog window for searching for something"""
7.
8.      def __init__(self,database):
9.          super().__init__()
10.
11.         self.database = database
12.
13.         #create widgets
14.         self.table_combo_box = QComboBox()
15.         self.search_box = QLineEdit()
16.         self.search_push_button = QPushButton("Search")
17.
18.         self.table_combo_box.addItem("Select Table")
19.         self.table_combo_box.addItem("MEMBERS")
20.         self.table_combo_box.addItem("PAYMENTS")
21.         self.table_combo_box.addItem("REGIME")
22.         self.table_combo_box.addItem("EXERCISE")
23.         self.search_box.setPlaceholderText("Enter Search Term")
24.
25.         #create layout
26.         self.layout = QVBoxLayout()
27.
28.         #add widgets to layout
29.         self.layout.addWidget(self.table_combo_box)
30.         self.layout.addWidget(self.search_box)
31.         self.layout.addWidget(self.search_push_button)
32.
```

```
33.         #set the window layout
34.         self.setLayout(self.layout)
35.         self.setWindowTitle("Search")
36.         self.setWindowIcon(QIcon("WindowIcon.png"))
37.
38.         #connections
39.         self.search_push_button.clicked.connect(self.search)
40.
41.
42.     def search(self):
43.             #method for searching the database
44.         results =
    searchQuery(self.database,self.table_combo_box.currentText(),""+self.search_box.text()+"")
45.         results_dialog = ResultsDialog(results)
46.         results_dialog.exec_()
```

Search Results Dialog

```
1.  from PyQt4.QtGui import *
2.
3.  class ResultsDialog(QDialog):
4.      """This class creates the dialog window for the search results"""
5.
6.      def __init__(self,searchResults):
7.          super().__init__()
8.
9.          self.searchResults = searchResults
10.
11.         #create widgets
12.         self.results = QTextEdit()
13.
14.         #create layout
15.         self.layout = QVBoxLayout()
16.
17.         #add widgets to layout
18.         self.layout.addWidget(self.results)
19.
20.         #set the window layout
21.         self.setLayout(self.layout)
22.         self.setWindowTitle("Results")
23.         self.setWindowIcon(QIcon("WindowIcon.png"))
24.
25.         self.searchResults = searchResults
26.         self.populateResultsBox()
27.
28.     def populateResultsBox(self):
29.             #method for populating the textEdit in the results dialog with the results from the
    search sql query
30.         self.string = ""
31.         for item in self.searchResults:
```

```
32.             self.string = ((self.string)+"\n\n"+str(item))
33.         self.results.setText(self.string)
```

Print Dialog

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtCore import *
3.  from gym_print_sql_function import *
4.  from gym_delete_function import *
5.
6.  class PrintDialog(QDialog):
7.      """This class creatres the dialog window for creating forms and printing them"""
8.
9.      def __init__(self,database):
10.         super().__init__()
11.
12.         self.database = database
13.
14.         #create widgets
15.         self.form_combo_box = QComboBox()
16.         self.item_select_combo_box = QComboBox()
17.         self.print_push_button = QPushButton("Print")
18.
19.         self.form_combo_box.addItem("Select Form")
20.         self.form_combo_box.addItem("Invoice")
21.         self.form_combo_box.addItem("Member Details")
22.         self.form_combo_box.addItem("Regime")
23.
24.
25.         #create layout
26.         self.layout = QVBoxLayout()
27.
28.         #add widgets to layout
29.         self.layout.addWidget(self.form_combo_box)
30.         self.layout.addWidget(self.item_select_combo_box)
31.         self.layout.addWidget(self.print_push_button)
32.
```

```
33.         #set the window layout
34.         self.setLayout(self.layout)
35.         self.setWindowTitle("Print")
36.         self.setWindowIcon(QIcon("Hugobells.png"))
37.
38.         #connections
39.         self.form_combo_box.currentIndexChanged.connect(self.itemComboBoxPopulate)
40.         self.print_push_button.clicked.connect(self.printInfo)
41.
42.     def printFunction(self,itemToPrint):
43.             #function that sends the textEdit to be printed and allows the user to select a printer
    through the print dialog
44.         dialog = QPrintDialog()
45.         if dialog.exec_() == QDialog.Accepted:
46.             itemToPrint.print_(dialog.printer())
47.
48.     def generateMemberInfo(self):
49.             #method that creates the textEdit containg the correct information for a member info
    print out
50.         columns
    =["MemberID","Name","Address","TelephoneNumber","MembershipType","InductionDate","Join
    Date","HowPaid","Amount","RegistrationFee","RegistrationDate","PaymentType","Comments"]
51.         sep = "."
52.         info = getMemberInfo(self.database,
    str(self.item_select_combo_box.currentText()).split(sep,1)[0])
53.         infoToPrint = QTextEdit()
54.         infoString = ""
55.         for item in info:
56.             count = 0
57.             for piece in item:
58.                 infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+" : "+str(piece)+"\n")
59.                 count += 1
60.         self.printFunction(infoToPrint)
```

```
61.
62.     def generateInvoice(self):
63.             #method that creates the textEdit containg the correct information for a print out of an
    invoice
64.         columns = ["MemberID","Payment Date","How Much","Paid"]
65.         sep = "."
66.         info,name = getInvoice(self.database,
    str(self.item_select_combo_box.currentText()).split(sep,1)[0])
67.         infoToPrint = QTextEdit()
68.         for list in name:
69.             for word in name:
70.                 for item in word:
71.                     infoToPrint.setText("Member Name : "+str(item)+"\n\n")
72.         for item in info:
73.             count = 0
74.             for piece in item:
75.                 infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+" : "+str(piece)+"\n")
76.                 count += 1
77.             infoToPrint.setText(infoToPrint.toPlainText()+"\n")
78.         self.printFunction(infoToPrint)
79.
80.     def generateRegime(self):
81.             #method that creates the textEdit containg the correct information for a members
    regime print out
82.         columns = ["MemberID","ExerciseID","Description","Start Date","End Date"]
83.         sep = "."
84.         info,name = getRegime(self.database,
    str(self.item_select_combo_box.currentText()).split(sep,1)[0])
85.         infoToPrint = QTextEdit()
86.         for list in name:
87.             for word in name:
88.                 for item in word:
89.                     infoToPrint.setText("Member Name : "+str(item)+"\n\n")
```

```
90.         for item in info:
91.             count = 0
92.             for piece in item:
93.                 infoToPrint.setText(infoToPrint.toPlainText()+columns[count]+" : "+str(piece)+"\n")
94.                 count += 1
95.             infoToPrint.setText(infoToPrint.toPlainText()+"\n")
96.         self.printFunction(infoToPrint)
97.
98.     def itemComboBoxPopulate(self):
99.         if self.form_combo_box.currentIndex() == 0:
100.                self.item_select_combo_box.clear()
101.                self.item_select_combo_box.addItem("Select Item")
102.            else:
103.                items = getItems(self.database, "MEMBERS")
104.                self.item_select_combo_box.clear()
105.                for count in range(len(items)):
106.                    self.item_select_combo_box.addItem(items[count])
107.
108.        def printInfo(self):
109.            if self.form_combo_box.currentIndex() == 1:
110.                self.generateInvoice()
111.            if self.form_combo_box.currentIndex() == 2:
112.                self.generateMemberInfo()
113.            if self.form_combo_box.currentIndex() == 3:
114.                self.generateRegime()
```

About Dialog

```
1.  from PyQt4.QtGui import *
2.  from PyQt4.QtCore import *
3.  import webbrowser
4.
5.  class AboutDialog(QDialog):
6.      """This is the about page"""
7.
8.      def __init__(self):
9.          super().__init__()
10.
11.         #create widgets
12.         self.text = QTextEdit()
13.         self.text.setPlainText("Created By Toby Kerslake \n\n\n\n User Manual available for
    download below")
14.         self.text.setReadOnly(True)#set the textEdit so the user can't rewrite the text
15.         self.userManualButton = QPushButton("User Manual")
16.
17.         #create layout
18.         self.layout = QVBoxLayout()
19.
20.
21.         #add widgets to layout
22.
23.         self.layout.addWidget(self.text)
24.         self.layout.addWidget(self.userManualButton)
25.
26.
27.         #set the window layout
28.         self.setLayout(self.layout)
29.         self.setWindowTitle("About")
30.         self.setWindowIcon(QIcon("WindowIcon.png"))
31.
```

```
32.         #connections
33.         self.userManualButton.clicked.connect(self.openUserManual)
34.
35.     def openUserManual(self):
36.             #method that opens up the user manual inside the users default browser
37.
    webbrowser.open('https://www.dropbox.com/s/61n4soosnvo1cnc/User%20Manual.docx?dl=0'
    )
```

Password Dialog

```
1.  from PyQt4.QtGui import *
2.
3.  class PasswordDialog(QDialog):
4.      """This class creates a dialog box for a password"""
5.
6.      def __init__(self):
7.          super().__init__()
8.
9.          #create widgets
10.         self.password_lineEdit = QLineEdit()
11.         self.password_lineEdit.setPlaceholderText("Password")
12.         self.password_lineEdit.setEchoMode(QLineEdit.Password)
13.         self.enterButton = QPushButton("Enter Password")
14.         self.closeButton = QPushButton("Close")
15.
16.         #create and set layout
17.         self.layoutMain = QVBoxLayout()
18.         self.layoutHorizontal = QHBoxLayout()
19.         self.layoutMain.addWidget(self.password_lineEdit)
20.         self.layoutHorizontal.addWidget(self.enterButton)
21.         self.layoutHorizontal.addWidget(self.closeButton)
22.         self.layoutMain.addLayout(self.layoutHorizontal)
23.         self.setLayout(self.layoutMain)
24.         self.setWindowTitle("Password")
25.         self.setWindowIcon(QIcon("WindowIcon.png"))
26.
27.         #connections
28.         self.enterButton.clicked.connect(self.close)
29.
30.     def close_method(self):
31.             #method for returning the entered password to check if it is correct by the main
    program file
```

```
32.                 return self.password_lineEdit.text()
```