Python SQLITE3 – Building a Quiz System

The aim of this tutorial is to take you through using Python to build a very simple quiz system. The complexity behind this is building it with a database running behind the scenes that will store the users details, their scores and the questions.

The tutorial will build the system step by step and has various points where you will have the choice to add your own complexity to the project. One example of this will be to look at working on the Graphical User Interface. Although this is covered in the tutorial, we will look at it only for the login screen.

Note:

- Sqlite3 comes as a standard library in all versions of Python from 2.5 onward
- This tutorial will require you to have some basic knowledge of SQL before starting
- The screenshots come from using PyCharm as my IDE. This tutorial will work using any Python IDE
- There is a set of youtube videos to accompany this tutorial, playlist can be found here (https://www.youtube.com/playlist?list=PLBRzFm0BKuhaQyJ37KiI9rg3R3CC6-v9e)
Contents:

Part 1 – Setting up the users and logging into the system

1.1 Using Python to make and connect to a SQLITE3 Database 3
1.2 Adding a table and inserting a user 3
1.3 Making a function to login to the 5
1.4 Making a function to add a user 7
1.5 Making a simple menu to run the functions 8

Part 2 – Planning the relational database 9

2.1 Planning the project and separating the files 10

Part 3 – Building the quiz

3.1 Inserting data into the tables 11
3.2 Setting up the quiz (building the menus) 11
3.3 Running the quiz 17
3.4 Creating more complex queries (Showing the users scores) 18
3.5 Presenting the users scores (Graphs using matplotlib) 19
3.6 Suggestions for extra functionality that you could add into this program 21

Part 4 – Creating a GUI for the login script using Tkinter

4.1 Creating the window 22
4.2 The Widgets and Layouts 23
4.3 Adding the title and textboxes 24
4.4 The submit and exit buttons 25
Part 1 – Setting up the users and logging into the system

1.1 – Using Python to make and connect to a SQLITE3 Database

The first thing we need to do is to make a new Python file and name it something descriptive as to what it’s going to do, which is to create a new database and make a user table. “Create Database” or “Create User Table” might be most suitable here.

One opened we will need to go ahead and import the sqlite3 library and make a connection to that database:

```python
import sqlite3

with sqlite3.connect("Quiz.db") as db:
    cursor = db.cursor()
```

If the database “Quiz.db” doesn’t exist then the program will make this database. If it does, this line will make a connection to the database that you can refer to/use later on by using the variable name “db”.

A cursor is a database object that is used to manipulate data and has commands associated with it such as “cursor.execute()” which will run a query, and “cursor.fetchall()” which collects the results of the query for you to use.

1.2 – Adding a table and inserting a user

The next step is to make our first table which is designed to store basic information about users.

```python
import sqlite3

with sqlite3.connect("Quiz.db") as db:
    cursor = db.cursor()

    CREATE TABLE IF NOT EXISTS user(
        userID INTEGER PRIMARY KEY,
        username VARCHAR(20) NOT NULL,
        firstname VARCHAR(20) NOT NULL,
        surname VARCHAR(20) NOT NULL,
        password VARCHAR(20) NOT NULL);
```

We use the “cursor.execute()” function for the first time here and within the brackets with enter in our SQL script as a string”.

Our table will be called “user” and will include the fields above. If you are familiar with mysql, we start noticing some differences in syntax here. You will notice that the Primary Key is declared next to the field. In sqlite3 you can declare this at the end of the SQL table creation statement by writing “INTEGER PRIMARY KEY” this will make the userID field not only the Primary Key but also it will automatically auto increment the value with each user that us created.
It's try to run the script. You will hopefully notice that a database named “Quiz.db” has been made in the same folder directory as the script.

Now that the table is all set up, we will try and create a test user to make sure that the table is accepting the data in the way in which we want it to. [https://youtu.be/omRXZhXihqQ](https://youtu.be/omRXZhXihqQ)

Again we’re going to use the “cursor.execute()” function to run this simple “INSERT INTO” script. This time however we are going to add a new line at the end called “db.commit()”. We use this here as it will save the data to the database. Without this the data would not be saved to the database.

```python
cursor.execute(""
INSERT INTO user(username,firstname,surname,password) VALUES("test_User","Bob","Smith","MrBob")
"")
db.commit()
```

Note that we haven’t needed to enter any data into the userID field as remember it is now autoincrement and a unique integer should be entered in here automatically every time we add a new user.

Let’s now run a script that will just do a quick test to check that the data has been entered into the database as we expected.

```python
cursor.execute("SELECT * FROM user")
print(cursor.fetchall())
```

```
C:\Python34\python.exe "C:\Users\pbgar\PycharmProjects\QuizProjectForDemo\Create a new database.py" 
[(1, 'test_User', 'Bob', 'Smith', 'MrBob')]
```

Process finished with exit code 0
1.3 – Making a function to login

https://youtu.be/ngynJQ0iVwM

Now that we have a user in our database, let’s make a function that allows our user to login using their credentials. In the form of a very basic algorithm, this is what we want to achieve:

1. Allow the user to enter their username and password
2. Connect to database and create a query which searches the database for the login credentials entered.
3. If a user has been found with the correct credentials, print a welcome message
4. If user not found, ask user if they would like to attempt to login again, otherwise exit system

To create the function we start with the command word “def” and give the function a name. As we’re not going to be passing in any details and everything is being handled inside the function we complete the line with empty brackets.

Part of the plan is to continue allowing the user to try again to login, I’ve started with an infinite loop “while True:”. This might be something you may want to change to only allow a maximum of three attempts to login and if this was the case, it might be a good idea to replace this line with “for i in range(3):”

```python
def login():
    while True:
        username = input("Enter your username: ")
        password = input("Enter your password:")
```

The next step is to make a connection to the database and construct a query that checks if the username and password exist.

```python
with sqlite3.connect("QuizScores.db") as db:
    cursor = db.cursor()
    find_user = ('SELECT * FROM users WHERE username = ? AND password = ?')
    cursor.execute(find_user, [(username),(password)]) # [] replaces the values of the ?
    results = cursor.fetchall()
```

Hopefully you now recognise most of the code above, in that we have made a connection to the database, constructed a query and executed that query. The new concept is that we have constructed a different type of query. The query needs to take in values (the username and the password). The problem here is that we can’t enter the variable names in as it would read them as strings, not the values being stored in the variables. You may be used to replacing variable values in a string as %s and declaring them later, but this method leaves us open to the potential of an SQL injection attack. Instead we use “?” where we need a variable inserting.

Notice on the next line where the cursor executes the query, we can at this point tell the database what to use instead of the question marks i.e `cursor.execute(find_user,[(username),(password)])`

We need to use the “cursor.fetchall()” function. Once a query is run, we need to tell the program to retrieve the results of this query.
Now we need to check if the query returned a value. If it has returned a value, we know the username and password has matched. To check this we need an IF statement "if results", which will simply check if there is a value stored in the variable results.

```python
if results:
    for i in results:
        print("Welcome "+i[2])
    return("exit")
else:
    print("Username and password not recognised")
    again = input("Do you want to retry?(Y/N)")
    if again.lower() == "n":
        print("Goodbye")
        time.sleep(1)
        return("exit")
```

The "else" allows the use the opportunity to exit the system or to try again.

Notice both functions return the string “exit”. We will use this later on when we run the functions.

```python
def login():
    while True:
        username = input("Enter your username: ")
        password = input("Enter your password:")
        with sqlite3.connect("Quizzes.db") as db:
            cursor = db.cursor()
            find_user = ('SELECT * FROM users WHERE username = ? AND password = ?')
            cursor.execute(find_user, [(username), (password)]) # [] replaces the values of the ?
            results = cursor.fetchall()

            if results:
                for i in results:
                    print("Welcome "+i[2])
                return("exit")
            else:
                print("Username and password not recognised")
                again = input("Do you want to retry?(Y/N)")
                if again.lower() == "n":
                    print("Goodbye")
                    time.sleep(1)
                    return("exit")
```

1.4 – Making a function to add a user

Now that we can log in to the database, it would be nice to have an option for new users to be able to create an account themselves rather than an administrator (i.e you) having to enter all users in yourself.

The first step is to ask the user to create a username. It’s important that we add some validation at this point to avoid someone entering in a username that already exists. The code below is using a lot of the concepts that we have learnt about already.

```python
def newUser():
    print("Add a new user")
    time.sleep(1)
#check is username is taken
found = 0
while found == 0:
    username = input("Enter a username: ")
    with sqlite3.connect("QuizScores.db") as db:
        cursor = db.cursor()
        find_user = ('SELECT * FROM users WHERE username = ?')
        cursor.execute(find_user,[(username)])
        if cursor.fetchall():
            print("Username Taken")
            else:
                found = 1
```

You may want to change the code above to allow the user the option to exit out of the system if they decide not to go ahead and create the account because they’re annoyed that their favourite username has been taken.

Now that we have checked that the username is free, we want to get the user to enter their details, do some more validation to check that they’ve entered the password correctly by getting them to enter their password twice and finally we want to execute a query to insert the values into the database.

```python
firstName = input("Please enter your first name: ")
surname = input("Please enter your last name: ")
password = input("Please enter a password: ")
password1 = input("Please re-enter your password: ")
while password != password1:
    print("Passwords did not match")
    password = input("Please enter a password: ")
    password1 = input("Please re-enter your password: ")

insertData = "'INSERT INTO users(username,firstname,surname,password) VALUES(?,?,?,?)'"
        cursor.execute(insertData,[(username),(firstName),(surname),(password)])
        db.commit() #save results to the database
```
1.5 – Making a simple menu to run the functions

To finish off this section we’re going to construct a simple menu system to allow the user the choice of running the functions that we have created.

```python
while True:
    print("Welcome to the system ")
    menu =(''
    1 - Create New User
    2 - Login
    3 - Exit \n'')

    userChoice = input(menu)

    if userChoice == "1":
        newUser()

    elif userChoice == "2":
        enter = login()
        if enter == "exit":
            break

    elif userChoice == "3":
        print("Goodbye")
        time.sleep(1)
        break

    else:
        print("Input not recognised, please try again")
```

The thing to point out with the code above is that it’s a “while True” loop meaning in theory it will continue to print the menu once the functions have been run, but we are also checking to see what the functions are returning. If the functions return the string “exit” the system breaks out of the loop and the program stops running.
Part 2 – Planning the relational database

The entity relationship diagram above represents the structure of the database that we are going to implement and uses a notation called “crows feet”. The relationships above are 1 to many, where the many is represented by the three prongs i.e a crows foot.

As an example of this, let’s look at the users table and the scores table. It is a “one to many” relationship as when a user is created, they are allocated a completely unique userID which will only ever appear in that table 1 time. The userID in the scores table is what is known as a foreign key, it will reference the userID from the users table. This part of the relationship is known as “many” as that userID can appear many times in the scores table i.e one user can take many quizzes and therefore can have many scores associated with their account.

The order in which we create these tables requires thought. If we created the scores table before the topics table, we won’t have anything to link topicID to in the scores table.

I strongly recommend at this stage putting the code below in a new Python file. As this is not part of the main functionality of your system (but is needed for the project) then you will want to separate it so that it doesn’t run everytime you run the system.

```
Topics Table

```
cursor.execute("
CREATE TABLE IF NOT EXISTS topics(
    topicID INTEGER PRIMARY KEY,
    topicName VARCHAR(20) NOT NULL);
"
)
```

```
Scores Table

```
cursor.execute(""
CREATE TABLE IF NOT EXISTS scores(
    scoreID INTEGER PRIMARY KEY,
    userID INTEGER NOT NULL ,
    score INTEGER NOT NULL,
    topicID INTEGER NOT NULL,
    FOREIGN KEY(userID) REFERENCES users(userID),
    FOREIGN KEY(topicID) REFERENCES topics(topicID));"
)
```

```
Questions Table

```
cursor.execute(""
CREATE TABLE IF NOT EXISTS questions(
    questionID INTEGER PRIMARY KEY,
    question VARCHAR(50),
    option1 VARCHAR(50),
    option2 VARCHAR(50),
    option3 VARCHAR(50),
    option4 VARCHAR(50),
    answer VARCHAR(50),
    FOREIGN KEY(topicID) REFERENCES topics(topicID));"
)
```

```
Testing Your Scripts

```
You may wish to run a script to verify that the tables have been created. If you’re used to using mysql this can be done with a simple “SHOW TABLES;” script. This is not the case with sqlite and you have to run the following script:

```

```
At this point it’s also worth looking at the flow of the system we are trying to build so that we have an overview of what the system will look like and do.

2.1 Planning the project and separating the files

As mentioned earlier in this section, rather than put all of the code in one big python file, it often makes sense in a project this size to separate out your functions into different files. This way it’s also easier to deal with part of the program in turn, it also makes it easier when returning to the project to update elements to find out where each that segment of the program is being handled and finally it makes it easier to deal with errors and debug your program (in my opinion).

You will see as you work through the rest of this tutorial that I use functions that are stored in different files. For example I have a quiz.py file that has a function in it called “quiz”. To access that function I need to import quiz and then to call it, I need to use quiz.quiz().

You don’t have to replicate the structure I have below, but it gives you an idea of how I have developed the project when building it.
Part 3 – Building the quiz

3.1 Inserting data into the tables

Inserting data into the database follows the same process of earlier, but we now need to consider the relationships in place. I have entered some sample data as you can see from the image below.

```python
with sqlite3.connect('QuizScores.db') as db:
    cursor = db.cursor()
    cursor.execute('''
        INSERT INTO topics (topicName)
        VALUES('Secondary Storage'), ('Networks'), ('System Software'), ('Ethical and Legal');
    ''
    db.commit()

    cursor.execute('''
        INSERT INTO questions (topicID,question,option1,option2,option3,option4,answer)
        VALUES('1','What type of storage is a Memory Stick?','Solid State','Magnetic','Optical','Volatile','1'),
        ('1','What type of storage is a CD?','Solid State','Magnetic','Optical','Volatile','3'),
        ('1','What type of storage is a Hard Disk Drive?','Solid State','Magnetic','Optical','Volatile','3'),
        ('1','What type of storage is a SSD?','Solid State','Magnetic','Optical','Volatile','1');
    ''
    db.commit()
```

If you come up with an error at this point, double check that foreign keys are in place. For example in the “questions” table there is the foreign key “topicID”, if I add a question to topicID 4, there obviously must be a topicID 4 in the “topics” table.

3.2 Setting up the quiz

Now that we have the database correctly structured, with data in the tables, we are ready to start planning how we want to present the quiz to the user in order for them to start using your system. Before we start, we need to go back to some of the code that we wrote in sections 1.3 (Log on function) and 1.5 (the menu function).

Starting with section 1.3. Once the user had logged in we made our function to return “exit”. As we don’t want our system to exit if correct login details have been entered, we want to return data that we will want to use in future functions. This will be the unique data that identifies the user who is logged in, the userID.

Go back to this function and change “if results” section to return – `return[i[0]]` :
Now for the menu section. Going back to the menu that runs the function. We need to tell the option to prompt the login function to do something once a value is returned from that function. I have changed the variable “enter” to “user” as it makes more sense going forward. The value returned from the login function will be stored under that variable.

Note from the code below:

- If the function doesn’t return the string “exit” it will run another function called `userMenu` which we have not yet created.
- The `userMenu` function takes a parameter which we are passing in the variable `user`. This again is the going to be the userID of the logged in user.

```python
elif userChoice == "2":
    user = Login.login()
    if user == "exit":
        break
    else:
        print("Starting Quiz")
        time.sleep(1)
        userMenu(user)
```
3.2.1 The User Menu

We need to make a user menu now, basically as we’ve told our previous menu to run one. The point of this menu is to:

- Allow the user a choice of which quiz topic that they might like to take
- An option to view scores
- An option to see a graph representing performance
- An option to exit/log off

There are two ways of making this next menu. We could type out all of our options, but this menu system would need updating every time we add a new topic. If you are an A-Level student you should be looking a dynamically creating this menu and therefore if a new topic is added then the menu system will reflect this. The dynamic method is a little harder to understand the logic and therefore you may for the time being decide to go for the easier concept of hard wiring in the next menu.

The dynamic method involved reading the topics into a list and then building a menu system based on this list. Compare both methods below.

<table>
<thead>
<tr>
<th>Hard Wired</th>
<th>Dynamically</th>
</tr>
</thead>
</table>
| def userMenu(user):
  while True:
    print("Welcome to the system ")
    menu = (''
      1 - Secondary Storage Quiz
      2 - Networks Quiz
      3 - Systems Software Quiz
      4 - Ethical and Legal Quiz
      5 - Show my scores
      6 - Graph
      7 - Exit 

    userChoice = input(menu)
    if userChoice == "1":
      quiz.quiz(user,1)
    elif userChoice == "2":
      quiz.quiz(user,2)
    else:
      print("Invalid choice")
  | def userMenu(user):
  with sqlite3.connect("QuizScores.db") as db:
    cursor = db.cursor()
    cursor.execute('SELECT * FROM topics;')
    results = cursor.fetchall()
    topics_menu = []
    for i in results:
      topics_menu.append(i[1])
    topics_menu.append("Show my scores")
    topics_menu.append("Graph")
    topics_menu.append("Exit")
    while True:
      menu_choice = 1
      for item in topics_menu:
        print(menu_choice,"",item)
        menu_choice +=1
      userChoice = input("Enter your choice: ")
      choices = len(topics_menu)-3
      if userChoice == 1:
        for item in topics_menu:
          if item == choices:
            print(item)
Hard Wired continued

Looking at the hard wired method, once a choice has been entered by the user, it will run a function called quiz. You can tell here that it is in a file named "quiz" where there is a function called "quiz" that takes two parameters, the user and the topicID so that the correct quiz can be run. The correct quiz is selected from the IF statements. There are 7 menu choices therefore there are 7 IF/ELIF statements to run each one, with an ELSE at the end just in case.

```python
userChoice = input(menu)

if userChoice == "1":
    quiz.quiz(user, 1)

elif userChoice == "2":
    quiz.quiz(user, 2)

elif userChoice == "3":
    quiz.quiz(user, 3)

elif userChoice == "4":
    quiz.quiz(user, 4)

elif userChoice == "5":
    stats.showScores(user)

elif userChoice == "6":
    stats.graph(user)

elif userChoice == "7":
    break
```
Dynamic continued

For the dynamic menu, we need to think a little more carefully about how to write the IF statements, as we can’t in essence, dynamically add endless IF statements depending on how many different quizzes there are to take. Instead the logic being applied here is:

- The user is going to enter a choice
- Find out the length of the list as this tell us how many menu choices there are
- We know the last three options are always going to be showing scores, graph and exit therefore we can deal with these first
- As long as the user enters a number that isn’t the last three choices then we can pass that option into the next quiz function

```python
while True:
    menu_choice = 1
    for item in topics_menu:
        print(menu_choice, "-", item)
        menu_choice += 1

    userChoice = input("Enter your choice: ")
    choices = len(topics_menu) - 3

    try:
        if userChoice == str(choices + 1):
            stats.showScores(user)
        elif userChoice == str(choices + 2):
            stats.graph(user)
        elif userChoice == str(choices + 3):
            break
        elif int(userChoice) <= choices:
            quiz.quiz(user, userChoice)
    except:
        print("Command not recognised")
```
3.2.2 Exception Handling

In the line `elif int(userChoice) <= choice` there is a chance that as we are converting the user input into an integer the user may have entered a value that can’t be converted into a string. This would normally throw up an error in Python. We want to avoid this as it isn’t helpful to the user and more importantly it will stop your system from working.

To avoid this problem we use exception handling where the system will “try” to run the code and it can’t and an “exception” is raised (i.e an error) then the we can instruct the program to do something other than stop running. In this case a simple message of “Command not recognised” is printed and as it is in a loop, the menu will be printed again.

This is quite a simple implementation of this type of error handling, and you can read more here:

https://wiki.python.org/moin/HandlingExceptions
3.3 Running the quiz

Before we build a function to run the quiz, it’s important to recap on a couple of key points from earlier:

1. The previous menu runs a function named “quiz” from a separate file named “quiz”. Therefore it’s important if you follow my structure from above that you now create a new file in the same directory of the quiz and call it “quiz.py”.

2. The next thing that we need to remember is the structure of the quiz table in the database and how we put the data in. Look back at the structure of your table to remind yourself. You should have found that we allowed a question with four possible answers and then an additional field to tell us which one of the four options was the correct one. All questions being structured this way helps us build a “one size fits all” function that will be suitable for all quizzes and all that is required is user and the topic that they have picked.

The code of this section is below. The logic is reasonably simple here and you will most likely be comfortable with all of the concepts used here by now.

- The function takes the user and the topic selected, does a search of the database of all of the questions from that topic and then loops through each question, presenting it to the user.
- The user then enters their answer and it checks if it is correct.
- A score is incremented as the user moves through the questions which is then used at the end to work out a percentage (this is used to keep all quiz scores consistent as topics might have varying numbers of questions)
- The userID, score and topic are then written to the scores table.

```python
import sqlite3, time

def quiz(userID, topicID):
    with sqlite3.connect("QuizScores.db") as db:
        cursor = db.cursor()
        score = 0
        cursor.execute("SELECT * FROM questions WHERE topicID=?", [{topicID}])
        questions = cursor.fetchall()
        numOfQuestions = 0 # used to help work out the score/percentage
        for question in questions:
            topic = question[1]
            print(question[2])
            print("1. %s \n 2. %s \n 3. %s \n 4. %s" % (question[3], question[4], question[5], question[6]))
            choice = input("Answer: ")
            if choice == question[7]:
                print("Correct")
                score += 1
                time.sleep(2)
                print(""")
            else:
                print("Incorrect")
            numOfQuestions+=1
        # works out percentage to keep all quiz scores consistent despite number of questions in topic
        score = int((score/numOfQuestions)*100)
        print("Your score was: ", score)
        # stores results of quiz in the scores table
        insertData="INSERT INTO scores(userID, score, topicID) VALUES(?,?,?)"
        cursor.execute(insertData, (userID, score, topicID))
        db.commit()
```
3.4 Creating more complex queries (Showing the users scores)

One of the functions on the user menu was an option to show scores from tests. At this point I will say that the query that we are about to build isn’t ideal as all it does is list out all scores of all tests ever taken. If you are building your own system you might think of better queries to do here or even write the data out to a file. The intention of this query is to demonstrate how to create queries that require taking the data from multiple tables. This is necessary as individual tables don’t store all of the data we would like to present to the user. For example the scores table holds information on all of the scores, but as well as the score it only stores the topicID and the userID, we need to know the name of the topics and the users and not just the number of its identifying field.

To know more about SQL queries and how they work, I can strongly recommend using the W3 Schools SQL interactive tutorials. [https://www.w3schools.com/sql/](https://www.w3schools.com/sql/)

The query below uses joins that link the tables on their primary and foreign keys. For future queries it’s important that you understand this concept. Use the W3 Schools link about to research “Inner Join”.

```python
def showScores(user):
    with sqlite3.connect("QuizScores.db") as db:
        cursor = db.cursor()

        query = "SELECT topics.topicName, scores.score, users.userID
        FROM users INNER JOIN topics INNER JOIN scores ON topics.topicID = scores.topicID ON users.userID = scores.userID
        WHERE ((users.userID)=?);"

        cursor.execute(query, [(user)])
        results = cursor.fetchall()
        for line in results:
            print(line[0], str(line[1]) + "")
```

The only other element of the query above is to explain the `cursor.execute` line of the query:

```python
cursor.execute(query, [(user)])
```

This time we are not just executing the query here, we are passing the user into the query as the last line of query requires it:

```python
WHERE ((users.userID)=?);"
```

**Challenges**

- Why not try and change this query to make it a little more user friendly, for example showing the scores grouped by the topic.
- Another suitable query would be to show the average score for each topic.
3.5 Presenting the users scores (Graphs using matplotlib)

This part of the project is a very simple intro into representing data as a graph. Presenting data this way can be helpful to the user and particularly in the context of this project where the user might want to compare scores to judge progress.

Matplotlib is not a standard python library and therefore will need to be installed. This can be done via PIP and my recommendation here is to use the free IDE PyCharm to add the package into your project in project settings. Some of students from previous years have avoided using matplotlib and instead chose to draw graphs using Tkinter or pygame. Although this is possible, matplotlib is a module designed specifically for creating graphs and therefore will always be more powerful and does have interactive features for the user as default, for example allowing the user to save the graph as a picture. You can read more about matplotlib here:

http://matplotlib.org/

Code:

```python
import sqlite3
import matplotlib.pyplot as plt

with sqlite3.connect("QuizScores.db") as db:
    cursor = db.cursor()
    y = []
    xaxis = []

    query = "SELECT topics.topicsName, scores.score, users.userID
            FROM users INNER JOIN (topics INNER JOIN scores ON topics.topicID = scores.topicID) ON users.userID = scores.userID
            WHERE ((users.userID)="{}")" "{}"

    for line in results:
        y.append(line[1]) #quiz score
        xaxis.append(line[0]) #topic name

plt.xticks(x, xaxis) #labels the x axis with the topic name
plt.bar(x, y)
plt.show()
```

Code Explained:

```
y = []
xaxis = []
```

We need to make two empty lists which are going to store the values for the X and Y axis. Each value holding a position in the list which we can iterate through later to create the graph.

```
for line in results:
    y.append(line[1]) #quiz score
    xaxis.append(line[0]) #topic name
```

This section is where we iterate through the results of the query to populate the lists we will use later for the graph (The quiz score for the y axis and the topic name for the x axis)
The `x` variable in the code above creates enough spaces for the names to be included on the x axis in another list called “x” and the basis of this is the number of positions there are in the y list so that the exact amount of spaces are created for the number of results that will be plotted.

```python
x = [i for i in range(len(y))]
plt.xticks(x, xaxis)  #labels the x axis with the topic name
```

Finally we need to tell the program to plot the values from x and y and you will see that we have chosen a “bar” chart here. There are other types of graphs that you can experiment with here, such as a line graph or a scatter graph. Again read the documentation further to get a better understanding of how to do this.

Now run the code and you should see a graph similar to the one below. If nothing appears, make sure that you have run this with a use that has taken at least one test already.

---

**Graph Interactivity**

One of the advantages of matplotlib is that the graphs are interactive as default and therefore when the graphs are plotted there are options for the user at the bottom of the graph that include moving the position, zooming in and out as well as being able to save the graphs as image files to a selected location.
3.6 Suggestions for extra functionality that you could add into this program

This project forms the skeleton of a project that would be suitable for A-Level. This is however just a skeleton project and other features should be considered. A requirement of the project is for your system to have a Graphical User Interface (GUI) and we will look at that briefly in the next section, but when starting your project it's worth considering the GUI right from the beginning.

Below are additional features that could be added to this project. Although this project should have more features to make it more complex and suitable for the end user, the features should always have your end user/stakeholders in mind.

- Admin login to see all users results
- Ability to add a quiz or edit questions & answers
- User to select results of specific quizzes to compare results
- Give feedback to user to suggest topics where performance has been worse
- Pick harder or easier quizzes or questions for the user based on past performance
Part 4 – Creating a GUI for the login script using Tkinter

In this section we’re going to look at the Tkinter library which comes built into Python. There are other ways to build a GUI and it might be worth also exploring guizero which is a simplified version (a wrapper) for the Tkinter library or you might look at Pygame which is increasingly popular and arguably more powerful than Tkinter. Both Pygame and guizero don’t come as standard into Python and therefore will require installing.

A good “getting started” tutorial for Tkinter can be found here:

http://usingpython.com/simple-gui-programming/

Tkinter and GUI design is a large topic area and something that will definitely need exploring further than this tutorial. This chapter goes through creating an interface for the “creating a new user” function covered in section 1.4 earlier in this tutorial.

The screenshot below shows that window that we are about to build so that you have an idea what we are working towards in this section.

![Create New User Window](image)
4.1 Creating the window

Firstly we need to setup the window that we are going to use. We do this by creating a TK object and storing it as a variable that we use quite a lot throughout this program, I have named mine newUserWindow. I will also be needing the set the background colour regularly therefore I have stored that in a variable so that if I decide to change the background colour later on then I only need to do this once and it will apply to all elements in the window.

```python
background='#C6D8F6'
newUserWindow = Tk()
newUserWindow.configure(background=background)
newUserWindow.geometry("430x300")
newUserWindow.mainloop()
```

The geometry of a window can be set as I have done so that the height and width are predefined, but if you can leave this blank and the window will automatically resize itself to fit the contents of the window.

Finally, the last line is important (newUserWindow.mainloop() ). All the lines of code relating to the contents within the window must be placed before the mainloop function. In short, the mainloop is needed to make everything run/display.

All future code for this window needs to be placed before the newUserWindow.mainloop() line of code.

4.2 The Widgets and Layouts

Before we start the next section we need to have a basic understanding of Widgets and the tkinter Layout manager options.

Widgets are elements that you can place on screen. Examples of widgets include labels (that display text and images), textboxes (that allow user entry) and buttons (allow actions to be carried out once the buttons are pressed/clicked on by the user). There are lots of other widgets other than the aforementioned, but these are the ones that we will work with for this part of the system.

Tkinter has different options for the layout of the widgets that we want to place on screen. The following website has a short but good description of the different options.

http://www.python-course.eu/tkinter_layout_management.php

The easiest to use is pack(). Pack will place the widgets in the next available position on the screen below one another. My classes often debate on whether or not this is a good option to use and they offer differ on option. I personally prefer the grid() option as it allows you to be more specific about where items appear on screen. The grid system allows you to see the screen as one big table with rows and columns in it and you can place your widgets inside these, almost like a spreadsheet.
4.3 Adding the title and textboxes

The code below shows the code needed to add the title at the top of the window and then the labels and textboxes for the user to enter their items into.

```python
Label(newUserWindow, text="Create New User", font="('Arial Narrow', 16)", bg=background).grid(row=0, column=0, columnspan=4)
username = Label(text="Please enter a username", bg=background).grid(row=2, column=1, sticky=W, pady=10)
usernameEnter = Entry()
usernameEnter.grid(row=2, column=2, columnspan=2, pady=10)
firstName = Label(text="Please enter your first name", bg=background).grid(row=3, column=1, sticky=W, pady=10)
firstNameEnter = Entry()
firstNameEnter.grid(row=3, column=2, columnspan=2, pady=10)
surname = Label(text="Please enter your surname", bg=background).grid(row=4, column=1, sticky=W, pady=10)
surnameEnter = Entry()
surnameEnter.grid(row=4, column=2, columnspan=2, pady=10)
password = Label(text="Please enter your password", bg=background).grid(row=5, column=1, sticky=W, pady=10)
passwordEnter = Entry(show="*")
passwordEnter.grid(row=5, column=2, columnspan=2, pady=10)
password2Enter = Entry(show="*")
password2Enter.grid(row=6, column=2, columnspan=2, pady=10)
```

A few points to consider from the code above:

- The label that is used to create the title is positioned at row 0 and column 0 but uses a command called "columnspan" which is the MS Excel equivalent of merging cells together.
- Pady and Padx allow for padding around the label. It’s worth using this to create a little space around the objects to reduce the chance of items being positioned too closely together.
- The passwordEnter entry box has a property called “show”. This will change the appearance of anything that the user enters into the box. Obviously you will want to hide what the user is typing in for their password from prying eyes.

4.4 The submit and exit buttons

```python
bottomFrame = Frame(newUserWindow, width=200, height=600)
bottomFrame.grid(row=7, column=1, padx=10, pady=2)
```

Because of the use of columns with the labels and textboxes above, it can be difficult to fit the buttons on the bottom of the page to make them the correct size and position. One way of handling this is to create a mini frame within the window that although fits inside the window, will act independently and therefore you can have a bit more freedom to position the buttons where you would like them. In the code above you will see that I have created a frame and defined where it should be and the height and width.

```python
Button(bottomFrame,text="Submit", command=insertData, width=10).grid(row=0, column=1)
Button(bottomFrame,text="Exit", command=lambda: close_user_window(newUserWindow), width=10).grid(row=0, column=2)
```

The purpose of a button is to be able to allow the user to activate when an event takes places, i.e when the user presses the button, it prompts a function to run. You will see that the “Submit” button has a “command” which in this case is set as “insertData”. I have a function written called insertData() and when you type in the function name, you will notice that you don’t need to include the brackets as the function takes not parameters.
The section button (“Exit”) runs a function called “close_user_window”. In this case however the function does take a parameter (which is newUserWindow and you may have guess that the function is written so that the window that you wish to close is passed into the function). To achieve this you need to use “lambda” which allows you in this instance to pass a value into the function, without it your system will produce an error.

Summary

This is now the end of the tutorial and I hope that you have found it useful. I also hope that it has allowed you to see the power behind creating a database in conjunction with Python code with the potential to create a dynamic and powerful program.

Mr Garside