# **Database Lab 3 - Introduction to Python for accessing SQLite**

**Objectives**

* To introduce some basic features of the Python API for accessing Databases
* To produce some small Python scripts that access SQLite database
* To provide confidence to learn more Python and further DB API features

**Background Knowledge**

By now you should have completed lab 1 and lab 2 and have some experience of basic SQL commands such as “CREATE”, “SELECT” and “INSERT”. This lab, you will learn how to query data in SQLite from Python.

We recommend you use Repl.it to practise these queries. However, if you wish to complete this worksheet on your own PC you will have to install Python version 3 and the sqlite3 python module. Details of how to install these pieces of software are beyond the scope of this tutorial, but can be easily found from many internet sites. A number of "dynamic tutorials" that demonstrate the installation of this software are provided in the software section of this website.

Note that this tutorial is not intended to be exhaustive, it is a simple introduction. You should ensure that you access the other resources available on the internet and in relevant textbooks to find out more.

**In this lab you will be,**

1. Creating a repository on GitHub

To create a blank github repository, log into GitHub, navigate to the Repositories tab and clicking on new.

Then, you can enter the repository name and click on create repository to create a blank repository. A public repository will allow others to view your repository, a private repository will require you to specify, by email or github username, who can access your repository. Use public for this module.

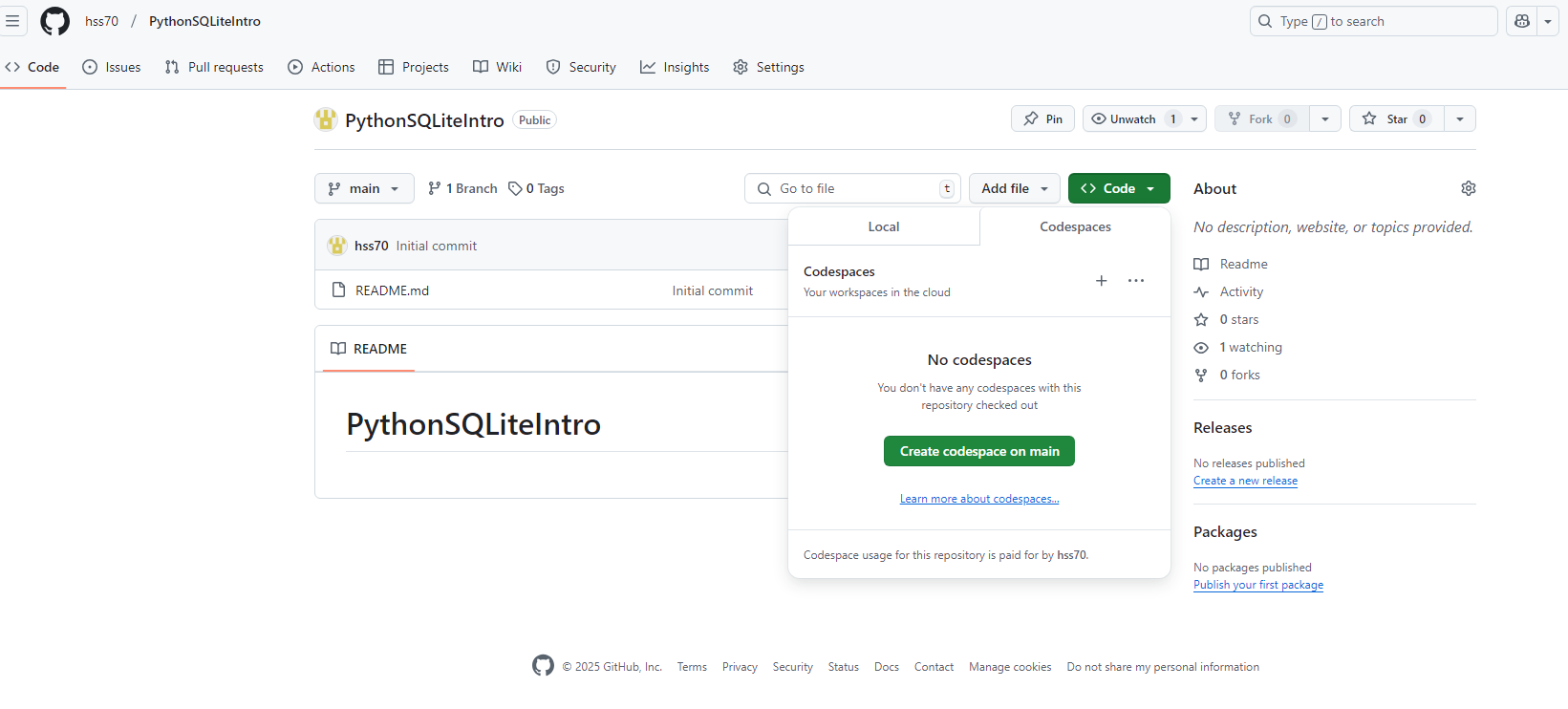
README files are useful for informing other developers about your project, any dependencies if they were to run it locally and to provide a guide on how to use your project. It is good practice to document your code in a README. Ensuring that Add a README file is checked will add a README file to your repository and commit a file to the main branch. A repository is not initiated until the first file is added via a commit.

A screenshot of a computer

Description automatically generated

1. Create your first GitHub CodeSpace

Once your repository has been created you will be taken to your repository page. Click on Code then “Create codespace on main” to create a codespace. You can only have 2 active codespaces at a time on the free tier, but you can close them and create them as you wish on your various repositories. For more info, look [here](https://github.blog/changelog/2022-11-09-codespaces-for-free-and-pro-accounts/).



Once your CodeSpace has been initialised, you can add a new file using the new file button which is the first icon:  
A close-up of a computer screen

Description automatically generated

Create a file called main.py

Once you start typing, a pop-up to install the python extension will show in the bottom right. Click install to install python in the codespace.

A screenshot of a computer

Description automatically generated

If you miss this pop-up, you can also install the python extension and other quality of life extensions in the extensions tab A screenshot of a computer

Description automatically generated

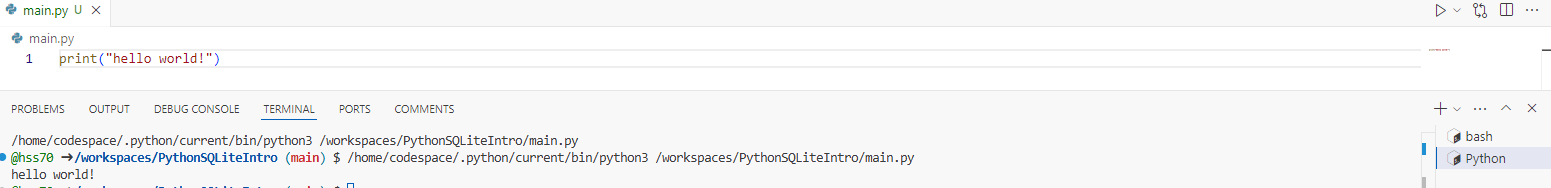
The CodeSpace creates a cloud hosted instance of Visual Studio Code, so most VS Code extensions will be here. This extension will add a play button in the top right corner which will allow you to run your python files.

1. The first script

The following is the typical "Hello World" program for Python. Use an editor to type it in as shown below;

print (“Hello World!”)

You should see a response text "Hello World” in the Terminal window.



You can also run your file in the terminal window by running python main.py



1. Committing and pushing to the repository

To save data in the repository you will need to commit files to the repository.

To do this, you will need to navigate to the source control tab A blue circle with white text

Description automatically generated

After this you will see your files with pending changes. Move your mouse over main.py and there will be a + button that you can use to stage your file changes.

Once you have staged your changes, you can write a meaningful commit message and click on commit to commit the changes.

A screenshot of a computer

Description automatically generated

This will add a commit to your local branch of your repository. You can see your local commits in the source control graph underneath

A screenshot of a message

Description automatically generated

Note how there is an origin/main and a main. This indicates that your changes are not in the online repository yet! To get them into the repository, you will need to push your commits to the remote branch using this push button 

This will push all your commits to the remote branch so they will be in the repository.

When you have multiple developers working on the same repo and same branch, you will need to use the fetch and pull buttons to bring in the latest changes from the repository and the relevant branches. This process is beyond the scope of this lab but you can find out more here:

[Beginner’s guide to GitHub: Adding code to your repository - The GitHub Blog](https://github.blog/developer-skills/github/beginners-guide-to-github-adding-code-to-your-repository/)

[Beginner’s guide to GitHub: Merging a pull request - The GitHub Blog](https://github.blog/developer-skills/github-education/beginners-guide-to-github-merging-a-pull-request/)

A legacy workflow used in the real world for git called Gitflow is a useful for understanding git fetch, git push and git pulls, as well as merges and branches. This workflow has fallen in popularity in recent years in favour of trunk based development

[Gitflow Workflow | Atlassian Git Tutorial](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

1. Connecting to a sqlite3 database

We are going to be using an example database in this session. Since we have learnt to use a Python script, we may as well use Python to set up the example database. Type (or cut-and-paste) the following the main.py file.

import sqlite3

conn = sqlite3.connect('store')

conn.execute("CREATE TABLE 'pet' (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), checkups SMALLINT UNSIGNED, birth DATE, death DATE)")

When you run the script, store.db file will be automatically created. You will not see any data in the database just yet.

You can view the db by installing the SQLite Viewer extension and opening the store file then clicking on “open anyway” and clicking on the SQLite viewer

A group of icons on a white background

Description automatically generated

Before moving on, take some time to have a look again at the script you have just run. Go through the script and put a comment above each statement that describes what you think the statement is doing. Feel free to post your code and comments in the “General Q&A” and compare what you think with another member of your class.

Try running the script more than one time. What did you get?

Now let’ try this script:

import sqlite3

conn = sqlite3.connect('store')

print ("Database has been created")

conn.execute("DROP TABLE IF EXISTS pet")

conn.execute("CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), checkups SMALLINT UNSIGNED, birth DATE, death DATE)")

print ("Table created successfully")

4. Inserting Data

Inserting data into a table in the database is not the simplest of the things you might want to do but is clearly fundamental to any database work with Python.

A new record is inserted into a table using the INSERT query. An example is the following:

"INSERT INTO pet VALUES (1, 'Fluffy', 'Alice', 'cat', 'f', 5, '2001-02-04', null)"

It is possible to specify which fields data will be inserted into too. Default values will be supplied in any fields not used, as long as these default values do not contravene the data description of the record.

"INSERT INTO pet (name, owner, species, birth) VALUES ('Fluffy', 'Alice', 'cat', '2001-02-04')"

Now let’s use this knowledge to insert some data. Consider the following Python script:

conn.execute("INSERT INTO pet (name,owner,species,sex,checkups,birth,death)VALUES \

('Fluffy','Harold','cat','f',5,'2001-02-04','')")

conn.execute("INSERT INTO pet (name,owner,species,sex,checkups,birth,death)VALUES \

('Claws','Gwen','cat','m',2,'2000-03-17','')")

conn.commit()

print("Records created successfully")

print("Total number of rows created :", conn.total\_changes)

What do you think will happen when the script is executed?

5. Reporting on Data

As well as being able to enter data, we need to be able to retrieve data from the table. This is relatively simple for single value queries, as illustrated in the following script:

You can see that a query that results in a list being returned. The data in the list in arranged in the same order as the columns of the table created earlier.

cursor = conn.execute("SELECT name,owner,species,sex,checkups,birth,death from pet")

for row in cursor:

print("name = ", row[0])

print("owner = ", row[1])

print("species = ", row[2])

print("sex = ", row[3])

print("checkups = ", row[4])

print("birth = ", row[5])

print("death = ", row[6], "\n")

**6. Updating data**

Now that you have seen how it is done, look up the UPDATE operation in the UPDATE syntax section of the SQLite Manual and provide a Python script that allows a death date to be entered for an animal identified by its name and owner's name

7. Further Work

We have really only started to look at how to use SQLite through Python sqlite3 module. There is much more to learn, and much more experience to be gained. You should ensure that you access the tutorials and books available to increase your Python skills and pick up good Python programming tips prior to tackling later tutorials. You will find that the practice you will get doing further lab work will help you refine the skills you have learnt.

We have been using the sqlite3 module in this tutorial.