

Python level 1

Introductory course Class 4

Fábio Neves & Luis Dias
September 18



PEA
Porto
Executive
Academy



AGENDA

Part 4 – Introduction to python

1. **Database**
 - a. Connection to database
 - b. Tables and Queries
 - c. SQL examples

Database



Database

SQLite

Python can be used in database applications, such as SQLite, MySQL (https://www.w3schools.com/python/python_mysql_getstarted.asp), MongoDB (https://www.w3schools.com/python/python_mongodb_getstarted.asp), etc.

In this class, we will learn how to integrate an **SQLite database with Python**. Nevertheless, the process for integrating other database applications is very similar.

Python SQLite3 module is used to integrate the SQLite database with Python. It provides a straightforward and simple-to-use interface for **interacting with SQLite databases**. There is no need to install this module separately, as it comes along with Python.

Database

Connection

Connecting to the SQLite Database can be established using the **connect() method**, passing the name of the database to be accessed as a parameter. If that database does not exist, then it'll be created.

```
import sqlite3  
sqliteConnection = sqlite3.connect('sql.db')
```

But what if you want to **execute some queries after the connection** is being made? For that, **a cursor has to be created using the cursor() method** on the connection instance, which will execute our SQL queries.

```
cursor = sqliteConnection.cursor()
```

The **SQL query to be executed** can be written **in the form of a string** and then executed by calling the **execute()** method on the cursor object. Then, the result can be fetched from the server by using the **fetchall()** method.

```
query = 'SQL query;  
cursor.execute(query)  
result = cursor.fetchall()
```

Database

Connection – Example



example.py

```
# Connect to DB and create a cursor
sqliteConnection = sqlite3.connect('sql.db')
cursor = sqliteConnection.cursor()

# Write a query and execute it with cursor
query = 'select sqlite_version();'
cursor.execute(query)

# Fetch and output result
result = cursor.fetchall()
print('SQLite Version is {}'.format(result))

# Always close the cursor when you do not need it anymore
cursor.close()
sqliteConnection.close()
print('SQLite Connection closed')
```

Result:

SQLite Version is [('3.31.1',)]
SQLite Connection closed

Database

Connection and Cursor

After the connection with the database is made, you can work with SQL to create, delete and insert tables. However, to query the table, the cursor object must be created.

The cursor is an object that is used to make the connection for executing SQL queries. It acts as middleware between SQLite database connection and SQL query. It is created after giving a connection to SQLite database.

In this website - <https://www.geeksforgeeks.org/python-sqlite/> - you will find a complete tutorial of python SQLite. In this class, we will see some examples, but all the functions that the SQLite module provides may be found on the above website.

Database

Queries: Select data from a table

`SELECT * FROM table_name;`

* means all the columns from the table
To select a specific column, replace * with the column name or the column names.

Example

Database:

| Email | Name | Score |
|------------------|-------|-------|
| geekk1@gmail.com | Geek1 | 25 |
| geekk2@gmail.com | Geek2 | 15 |
| geekk3@gmail.com | Geek3 | 36 |
| geekk4@gmail.com | Geek4 | 27 |
| geekk5@gmail.com | Geek5 | 40 |
| geekk6@gmail.com | Geek6 | 14 |
| geekk7@gmail.com | Geek7 | 10 |



example.py

```
import sqlite3

# Connecting to sqlite
# connection object
connection_obj = sqlite3.connect('geek.db')

# cursor object
cursor_obj = connection_obj.cursor()

# to select all columns we will use
statement = "SELECT * FROM GEEK"

cursor_obj.execute(statement)

print("All the data")
output = cursor_obj.fetchall() # fetch all rows
for row in output:
    print(row)

connection_obj.commit()

# Close the connection
connection_obj.close()
```


Database

Queries: Select data from a table

Example
Database:

| Email | Name | Score |
|------------------|-------|-------|
| geekk1@gmail.com | Geek1 | 25 |
| geekk2@gmail.com | Geek2 | 15 |
| geekk3@gmail.com | Geek3 | 36 |
| geekk4@gmail.com | Geek4 | 27 |
| geekk5@gmail.com | Geek5 | 40 |
| geekk6@gmail.com | Geek6 | 14 |
| geekk7@gmail.com | Geek7 | 10 |



example.py

```
import sqlite3

# Connecting to sqlite
# connection object
connection_obj = sqlite3.connect('geek.db')

# cursor object
cursor_obj = connection_obj.cursor()

# to select all columns we will use
statement = "SELECT * FROM GEEK"

cursor_obj.execute(statement)

print("Limited data")
output = cursor_obj.fetchmany(5) # fetch only 5 rows
for row in output:
    print(row)

connection_obj.commit()

# Close the connection
connection_obj.close()
```

Database

Queries: Where clause



example.py

```
SELECT column_1, column_2,...,column_N
```

```
FROM table_name
```

```
WHERE [search_condition]
```

Here, in this [search_condition], you can use comparison or logical operators to specify conditions.

For example: =, >, <, !=, LIKE, NOT, etc...

Example
Database:

| Student_ID | First_Name | Last_Name | Age | Department |
|------------|------------|-----------|-----|------------|
| 1 | Rohit | Pathak | 21 | IT |
| 2 | Nitin | Biradar | 21 | IT |
| 3 | Virat | Kohli | 30 | CIVIL |
| 4 | Rohit | Sharma | 32 | COMP |

Database

Queries: Where clause

Example
Database:

| Student_ID | First_Name | Last_Name | Age | Department |
|------------|------------|-----------|-----|------------|
| 1 | Rohit | Pathak | 21 | IT |
| 2 | Nitin | Biradar | 21 | IT |
| 3 | Virat | Kohli | 30 | CIVIL |
| 4 | Rohit | Sharma | 32 | COMP |



example.py

```
import sqlite3

connection = sqlite3.connect('geeksforgeeks_student.db')
cursor = connection.cursor()

# WHERE clause to retrieve data
cursor.execute("SELECT * FROM STUDENT WHERE Department = 'IT'")

# printing the cursor data
print(cursor.fetchall())

connection.commit()
connection.close()
```

Database

Full example: creating, inserting values and query tables



example.py

```
# importing sqlite3 module
import sqlite3

# create connection by using object
# to connect with hotel_data database
connection = sqlite3.connect('hotel_data.db')

# query to create a table named hotel
connection.execute(""" CREATE TABLE hotel
                    (FIND      INT PRIMARY KEY   NOT NULL,
                     FNAME     TEXT              NOT NULL,
                     COST      INT               NOT NULL,
                     WEIGHT    INT); """)

# insert query to insert food details in the above table
connection.execute("INSERT INTO hotel VALUES (1, 'cakes', 800, 10 )")
connection.execute("INSERT INTO hotel VALUES (2, 'biscuits', 100, 20 )")
connection.execute("INSERT INTO hotel VALUES (3, 'chocos', 1000, 30 )")

print("All data in hotel table\n")

# create a cursor object for select query
cursor = connection.execute("SELECT * from hotel ")

# display all data from hotel table
for row in cursor:
    print(row)
```

Result:

All data in food table
(1, 'cakes', 800, 10)
(2, 'biscuits', 100, 20)
(3, 'chocos', 1000, 30)

Database

Full example: inserting values and query tables



example.py

```
# importing sqlite3 module
import sqlite3

# create connection by using object
# to connect with hotel_data database
connection = sqlite3.connect('hotel_data.db')

# insert query to insert food details
# in the above table
connection.execute("INSERT INTO hotel VALUES (1, 'cakes', 800, 10 )")
connection.execute("INSERT INTO hotel VALUES (2, 'biscuits', 100, 20 )")
connection.execute("INSERT INTO hotel VALUES (3, 'chocos', 1000, 30 )")

print("Food id and Food Name\n")

# create a cursor object for select query
cursor = connection.execute("SELECT FIND, FNAME from hotel ")

# display all data from hotel table
for row in cursor:
    print(row)
```

Result:

Food id and Food Name
(1, 'cakes')
(2, 'biscuits')
(3, 'chocos')

Database

SQLite database to pandas dataframe

Databases are very useful for storing the data, but pandas dataframes are very helpful for analysing, cleaning and plotting the data. As so, after connecting to the database, we may convert any table to a pandas dataframe:



example.py

```
import pandas as pd
import sqlite3

# Connecting to sqlite
conn = sqlite3.connect('database.db')

# Creating a cursor object using the cursor() method
cursor = conn.cursor()

# Keep the table in dataframe
df = pd.read_sql_query("SELECT * FROM table_name", conn)
```

Database

SQL examples

Consider the following tables from a database called **shop.db**.

Customers

| id | name | city |
|----|-------|--------|
| 1 | Ana | Porto |
| 2 | Bruno | Lisboa |
| 3 | Clara | Porto |

Orders

| id | customer_id | amount |
|----|-------------|--------|
| 1 | 1 | 25.0 |
| 2 | 1 | 40.0 |
| 3 | 2 | 15.0 |
| 4 | 3 | 60.0 |
| 5 | 3 | 10.0 |

Database

SQL examples – Get all customers who live in Porto



example.py

```
import sqlite3

con = sqlite3.connect("shop.db")
cur = con.cursor()

cur.execute("SELECT id, name, city FROM customers WHERE city = 'Porto'")

for row in cur.fetchall():
    print(row) # (id, name, city)

con.close()
```


Database

SQL examples – Show orders that are either over 20 or belong to customer_id = 2, sorted by amount descending.



example.py

```
import sqlite3

con = sqlite3.connect("shop.db")
cur = con.cursor()

cur.execute("""
    SELECT id, customer_id, amount
    FROM orders
    WHERE amount > 20 OR customer_id = 2
    ORDER BY amount DESC""")

for row in cur.fetchall():
    print(row) # (order_id, customer_id, amount)

con.close()
```

Database

SQL examples – For each customer, show their name and total amount spent. We'll exclude customers with no orders using an INNER JOIN.



example.py

```
import sqlite3

con = sqlite3.connect("shop.db")
cur = con.cursor()

cur.execute("""
    SELECT c.name, SUM(o.amount) AS total_spent
    FROM customers AS c
    INNER JOIN orders AS o
    ON o.customer_id = c.id
    GROUP BY c.id, c.name
    ORDER BY total_spent DESC""")

for row in cur.fetchall():
    print(row) # (name, total_spent)

con.close()
```



Class Wrap-up



What have we learned today?

- Learned to use **Python** to connect with **Databases**
- Leverage **Python** and **SQL** to **query** tables in Databases
- Got familiar with simple **SQL** examples to explore and retrieve data



**PRACTICE
PRACTICE
PRACTICE**



Exercises for today

The screenshot displays the GitHub interface for the repository 'natix-python-level-1'. The repository is private and has 0 watches, 0 forks, and 0 stars. The main branch is 'main'. A commit by 'luis.dias@tui.com' is shown, with a file tree listing 'class material', 'exercises', and 'README.md'. The README file is open, showing the title 'Natix Python Level 1' and a 'Course Overview' section. The course overview states: 'This introductory course is designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.' Below this is the 'Course Structure' section, which includes 'Class 1: Introduction to Python - The Basics' with sub-points: 'Installation & Setup', 'Python installation', and 'Setting working directory'. The right sidebar shows repository statistics: 0 stars, 0 watching, and 0 forks. It also includes sections for 'Releases' (no releases published), 'Packages' (no packages published), and 'Languages' (Python 100.0%).

natix-python-level-1 Private

Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file Add file Code

luis.dias@tui.com Revise course title in README.md for clarity and consistency 32ca41e · 1 minute ago 6 Commits

- class material Add initial exercises and solutions for Class 1, including a ... 20 minutes ago
- exercises Add initial exercises and solutions for Class 1, including a ... 20 minutes ago
- README.md Revise course title in README.md for clarity and consiste... 1 minute ago

README

Natix Python Level 1

Course Overview

This introductory course is designed for beginners to learn how to use Python as a powerful tool for working with data. Students will gain a solid foundation in Python programming while focusing on practical skills for accessing, manipulating, and analyzing data effectively.

Course Structure

Class 1: Introduction to Python - The Basics

- Installation & Setup
 - Python installation
 - Setting working directory

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages

Python 100.0%

Link to exercises: https://github.com/diaxz12/natix-python-level-1/blob/main/exercises/Class4_exercises.py



THANK YOU 😊

Questions?

Fábio Neves



fnssm26@gmail.com

Luis Dias



luis.f.s.m.dias@gmail.com