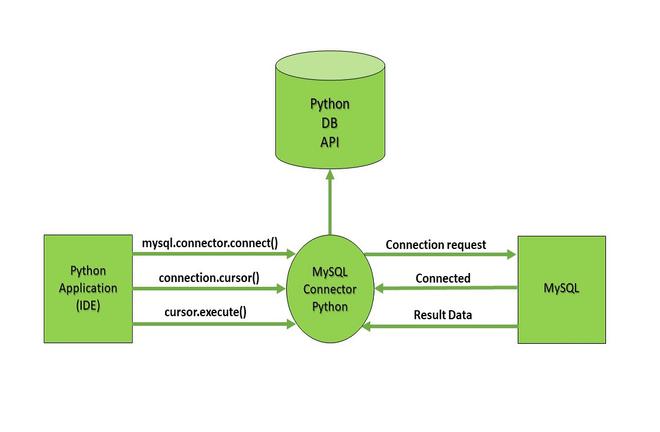
**How to Connect Python with SQL Database?**

Python is a high-level, general-purpose, and very popular programming language. Basically, it was designed with an emphasis on code readability, and programmers can express their concepts in fewer lines of code. We can also use Python with SQL. In this article, we will learn how to connect SQL with Python using the ‘MySQL Connector Python module. The diagram given below illustrates how a connection request is sent to MySQL connector Python, how it gets accepted from the database and how the cursor is executed with result data.



*SQL connection with Python*

**Connecting MySQL with Python**

To create a connection between the MySQL database and Python, the **connect()** method of **mysql.connector** module is used. We pass the database details like HostName, username, and the password in the method call, and then the method returns the connection object.

The following steps are required to connect SQL with Python:

**Step 1:** Download and Install the free MySQL database from [here.](https://www.mysql.com/downloads/)

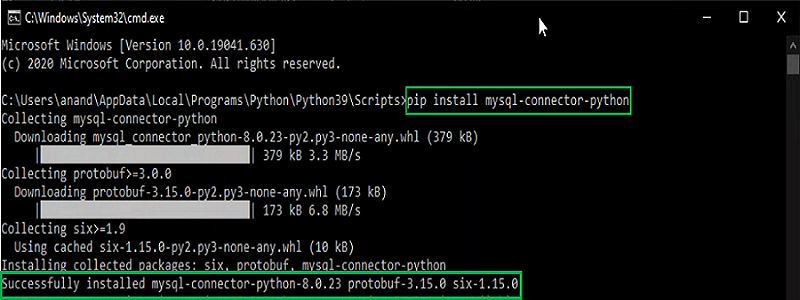
**Step 2:**After installing the MySQL database, open your Command prompt.

**Step 3:** Navigate your Command prompt to the location of PIP. Click here to see,[How to install PIP?](https://www.geeksforgeeks.org/how-to-install-pip-on-windows/)

**Step 4:** Now run the commands given below to download and install “MySQL Connector”. Here, mysql.connector statement will help you to communicate with the MySQL database.

**Download and install “MySQL Connector”**

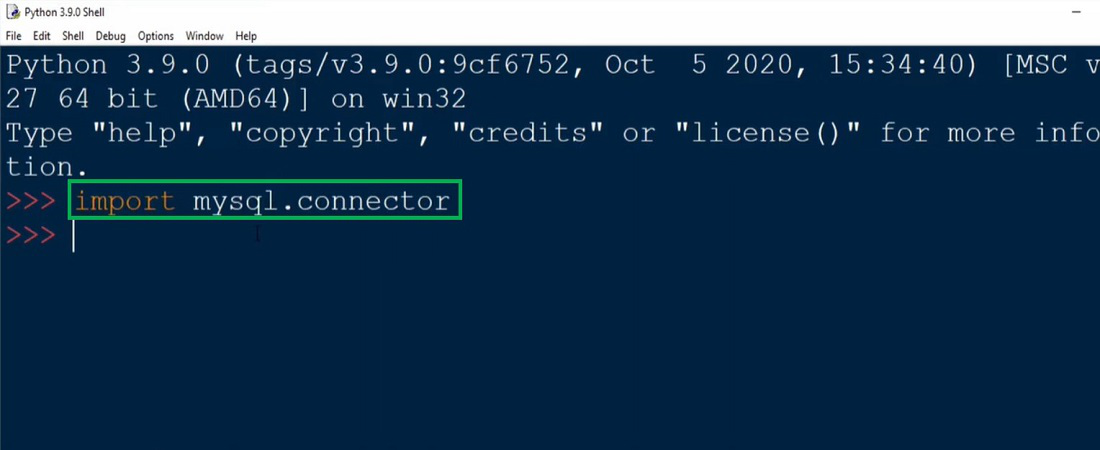
pip install mysql-connector-python



**Step 5: Test MySQL Connector**

To check if the installation was successful, or if you already installed “MySQL Connector”, go to your IDE and run the given below code :

import mysql.connector



If the above code gets executed with no errors, “MySQL Connector” is ready to be used.

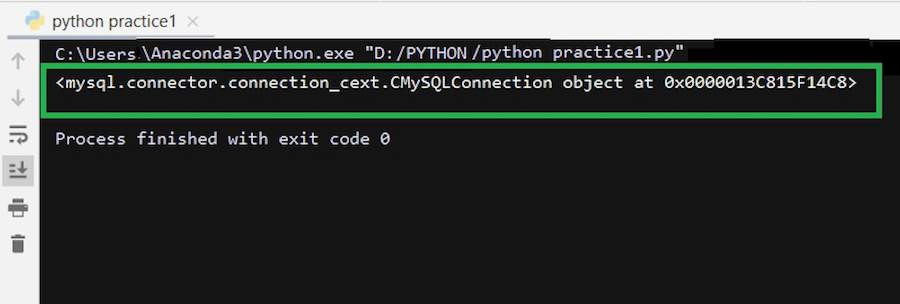
**Step 6: Create Connection**

Now to connect SQL with Python, run the code given below in your IDE.

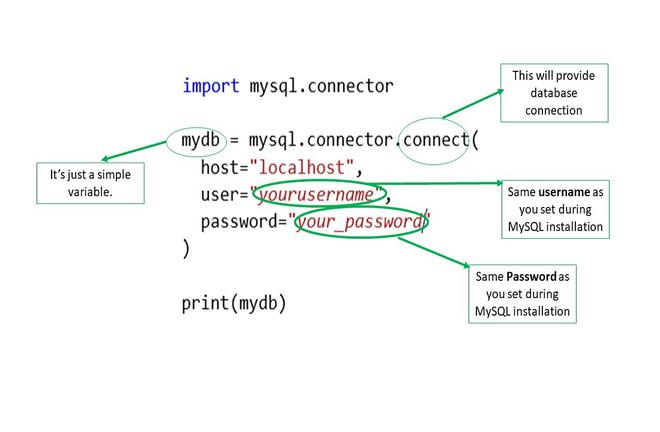
* Python3

|  |
| --- |
| # Importing module  **import** mysql.connector    # Creating connection object  mydb **=** mysql.connector.connect(      host **=** "localhost",      user **=** "yourusername",      password **=** "your\_password"  )    # Printing the connection object  print(mydb) |

**Output:**



**Here, in the above code:**



*Code Info*

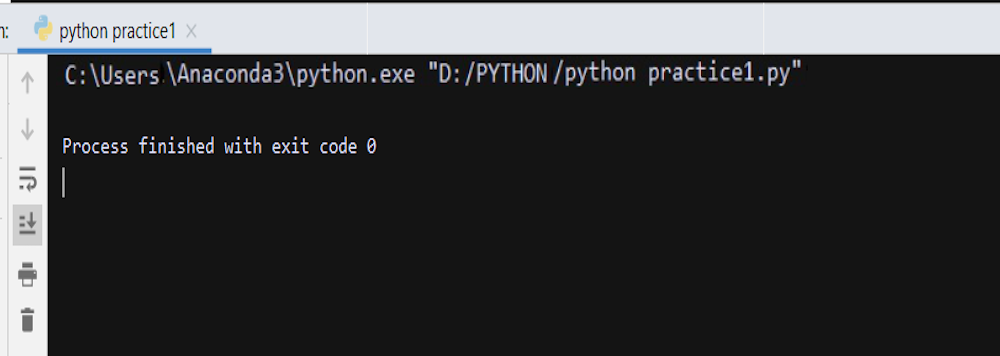
**Creating MySQL Database**

To create a database, we will use CREATE DATABASE database\_name statement and we will execute this statement by creating an instance of the ‘cursor’ class.

* Python3

|  |
| --- |
| **import** mysql.connector    mydb **=** mysql.connector.connect(      host **=** "localhost",      user **=** "yourusername",      password **=** "your\_password"  )    # Creating an instance of 'cursor' class  # which is used to execute the 'SQL'  # statements in 'Python'  cursor **=** mydb.cursor()    # Creating a database with a name  # 'geeksforgeeks' execute() method  # is used to compile a SQL statement  # below statement is used to create  # the 'geeksforgeeks' database  cursor.execute("CREATE DATABASE geeksforgeeks") |

**Output:**

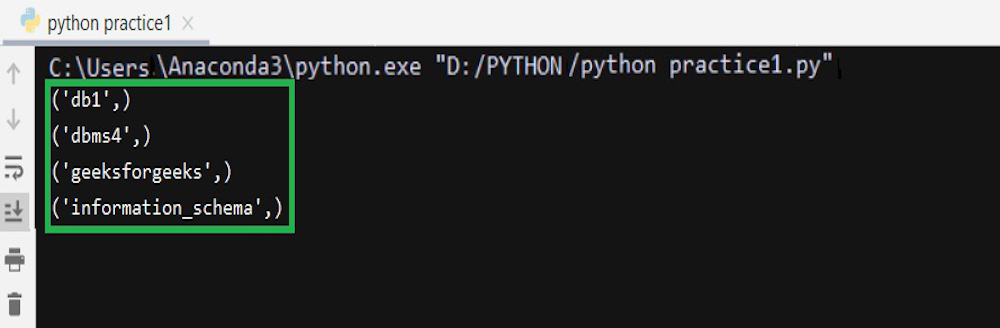


If the database with the name ‘geeksforgeeks’ already exists then you will get an error, otherwise no error. So make sure that the new database that you are creating does not have the same name as the database already you created or exists previously. Now to check the databases that you created, use*“SHOW DATABASES” – SQL statement*i.e. cursor.execute(“SHOW DATABASES”)

* Python3

|  |
| --- |
| **import** mysql.connector    mydb **=** mysql.connector.connect(      host **=** "localhost",      user **=** "root",      password **=** "1234"  )    # Creating an instance of 'cursor' class  # which is used to execute the 'SQL'  # statements in 'Python'  cursor **=** mydb.cursor()    # Show database  cursor.execute("SHOW DATABASE")    **for** x **in** cursor:  **print**(x) |

**Output:**



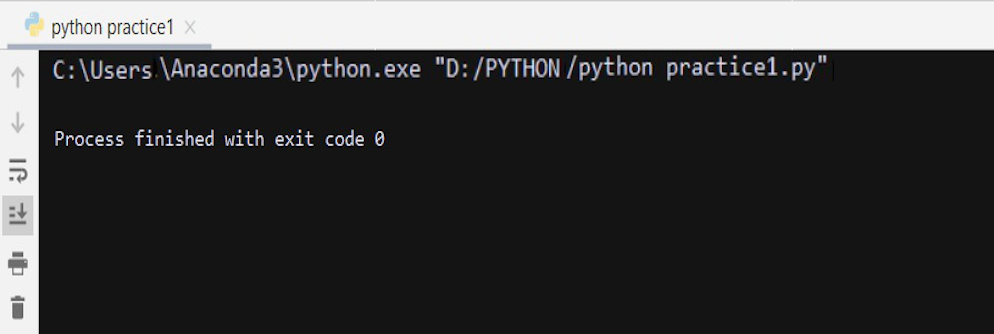
**Creating Tables**

Now to create tables in a database, first, we have to select a database and for that, we will pass *database = “NameofDatabase”* as your fourth parameter in connect() function. Since we have created a database with the name ‘geekforgeeks’ above, so we will use that and create our tables. We will use *CREATE TABLE gfg (variableName1 datatype, variableName2 datatype)* statement to create our table with the name ‘gfg’.

* Python3

|  |
| --- |
| **import** mysql.connector    mydb **=** mysql.connector.connect(      host **=** "localhost",      user **=** "yourusername",      password **=** "your\_password",      database **=** "geeksforgeeks"  )    cursor **=** mydb.cursor()    # Creating a table called 'gfg' in the  # 'geeksforgeeks' database  cursor.execute("CREATE TABLE gfg (name VARCHAR(255), user\_name VARCHAR(255))") |

**Output:**

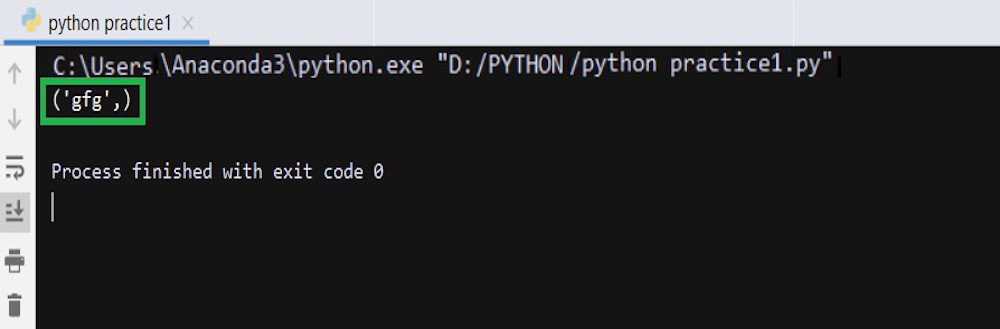


If the table with the name ‘gfg’ already exists, you will get an error, otherwise no error. So make sure that the new table that you are creating does not have the same name as the table already you created or exists previously. Now to check tables that you created, use *“SHOW TABLES” –**SQL statement* i.e. cursor.execute(“SHOW TABLES”).

* Python3

|  |
| --- |
| **import** mysql.connector    mydb **=** mysql.connector.connect(      host **=** "localhost",      user **=** "root      password **=** "1234",      database **=** "geeksforgeeks"  )    cursor **=** mydb.cursor()    # Show existing tables  cursor.execute("SHOW TABLES")    **for** x **in** cursor:    print(x) |

**Output:**



**Notes:**

* **mysql.connector**allows Python programs to access MySQL databases.
* **connect()** method of the MySQL Connector class with the arguments will connect to MySQL and would return a MySQLConnection object if the connection is established successfully.
* **user = “yourusername”**, here “yourusername” should be the same username as you set during MySQL installation.
* **password = “your\_password”**, here “your\_password” should be the same password as you set during MySQL installation.
* **cursor()**is used to execute the SQL statements in Python.
* **execute()** method is used to compile a SQL statement.

Connecting Python with an SQL database offers several advantages, making it a popular choice for data manipulation, analysis, and application development. Here are some key advantages:

1. Data Retrieval and Manipulation: Python can be used to retrieve data from an SQL database, allowing you to perform various data manipulation tasks such as filtering, sorting, aggregating, and transforming data using Python's rich libraries and tools.

2. Data Analysis and Visualization: Python has powerful libraries like Pandas, NumPy, and Matplotlib that enable you to perform advanced data analysis and create informative visualizations based on the data stored in your SQL database.

3. Integration with Other Tools: Python can easily integrate with other data science and machine learning tools, making it a valuable part of the data analysis and machine learning pipeline. You can use SQL to extract and prepare data and then use Python for modeling and analysis.

4. Automation and Scripting: Python allows you to automate repetitive database tasks and create scripts for data extraction, transformation, and loading (ETL) processes, making it easier to maintain and update your data.

5. Web Development: Python can be used for web development, and connecting to an SQL database is essential for building dynamic web applications that require data storage and retrieval.

6. Data Import and Export: Python provides modules like `csv`, `Excel`, and others to easily import and export data between SQL databases and various file formats, facilitating data migration and sharing.

7. Cross-Platform Compatibility: Python is a cross-platform language, which means you can use it to connect to SQL databases on different operating systems, ensuring flexibility and portability.

8. Extensive Libraries: Python has numerous database connectors and ORMs (Object-Relational Mapping tools) such as SQLAlchemy, Django ORM, and more, which simplify database interactions and improve code readability.

9. Scalability: Python can handle large datasets and complex database operations efficiently, making it suitable for both small-scale and large-scale applications.

10. Community Support: Python has a vast and active community, so you can easily find help, documentation, and resources when working with Python and SQL databases.

11. Security: When used correctly, Python and SQL databases can be configured with robust security measures to protect sensitive data and prevent unauthorized access.

Overall, connecting Python with an SQL database enhances the capabilities of both technologies and allows you to leverage Python's flexibility and extensive libraries for working with your data. It is a powerful combination for various data-related tasks, from data analysis and reporting to web development and automation.