# **Python MySQL Database Connection Explained with Examples­­**

This document will demonstrate how to perform **MySQL database connectivity in Python using simple 5 steps**.

**Goals of this lesson,**  in this lesson, you’ll learn:­

* How to connect MySQL Server and create a table in MySQL from Python
* Different MySQL Connection arguments we can use to connect to MySQL
* How to change the MySQL connection timeout when connecting through Python

Note: We are using **MySQL Connector Python** to connect MySQL.

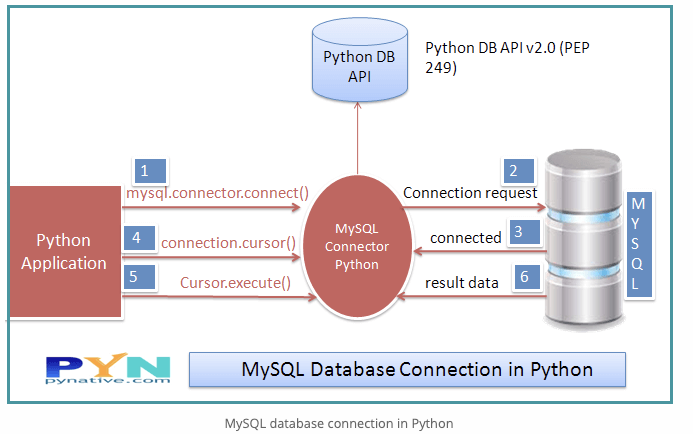
**Arguments required to connect MySQL from Python**

You need to know the following detail of the MySQL server to perform the connection from Python.

* **Username** – i.e., the username that you use to work with MySQL Server. The default username for the MySQL database is a **root**
* **Password** – Password is given by the user at the time of installing the MySQL database. If you are using root then you won’t need the password.
* **Host Name** – is the server name or Ip address on which MySQL is running. if you are running on localhost, then you can use localhost, or it’s IP, i.e. 127.0.0.0
* **Database Name** – Database name to which you want to connect. Here we are using Database named ‘**Electronics**‘ because we have already created this for our example.

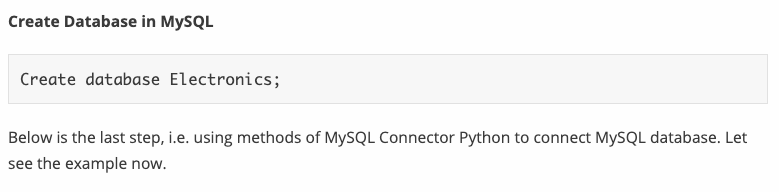
## **Steps to connect MySQL database in Python using MySQL Connector Python**

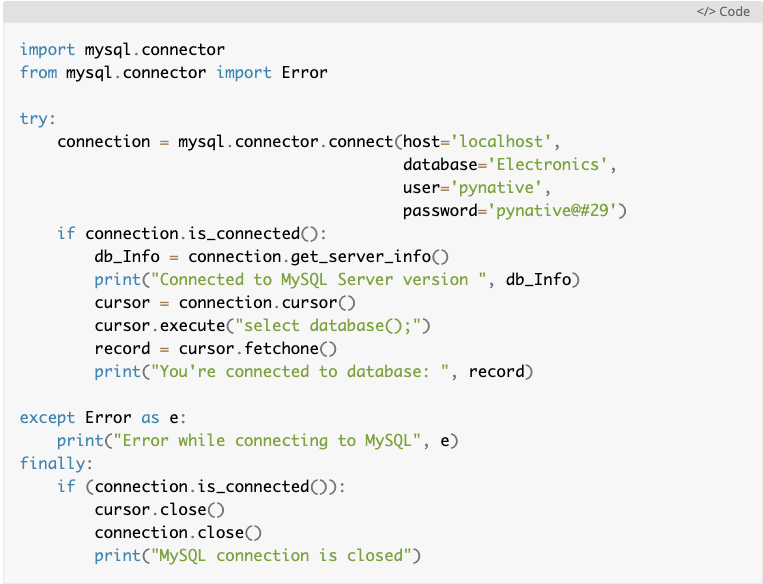
* [Install MySQL Connector Python using pip](https://pynative.com/install-mysql-connector-python/).
* Use the **mysql.connector.connect()** method of MySQL Connector Python with required parameters to connect MySQL.
* Use the connection object returned by a connect() method to create a **cursor** object to perform Database Operations.
* The **cursor.execute()** to execute SQL queries from Python.
* Close the Cursor object using a **cursor.close()** and MySQL database connection using **connection.close()** after your work completes.
* Catch Exception if any that may occur during this process.



### **Python Example to connect MySQL Database**

To connect the MySQL database, you must know the database name you want to connect. Run below query on the MySQL console if you have not created any database in MySQL. Otherwise, you can skip the below query.





After connecting to MySQL Server, you should get the below output.

**Understand the Python MySQL Database connection program**

**import mysql.connector**

* This line imports the MySQL Connector Python module in your program so you can use this module’s API to connect MySQL.

**from mysql.connector import Error**

* **mysql connector Error object** is used to show us an error when we failed to connect Databases or if any other database error occurred while working with the database. Example ACCESS DENIED ERROR when username or password is wrong.

**mysql.connector.connect()**

* Using this method we can connect the MySQL Database, this method accepts four required parameters: **Host, Database, User and Password** that we already discussed.
* **connect()** method established a connection to the MySQL database from Python application and returned a MySQLConnection object. Then we can use MySQLConnection object to perform various operations on the MySQL Database.
* The **Connect()** method can throw an exception, i.e. Database error if one of the required parameters is wrong. For example, if you provide a database name that is not present in MySQL, then the Python application throws an exception. So check the arguments that you are passing to this method.

**connection.is\_connected()**

* **is\_connected() is the method of the MySQLConnection class through which we can verify our python application is connected to MySQL.**

**connection.cursor()**

* **This method returns a cursor object. Using a cursor object, we can execute SQL queries.**
* **The MySQLCursor class instantiates objects that can execute operations such as SQL statements.  
  Cursor objects interact with the MySQL server using a MySQLConnection object.**

**cursor.close()**

* **Using the cursor’s close method we can close the cursor object. Once we close the cursor object, we can not execute any SQL statement.**

**connection.close()**

* **At last, we are closing the MySQL database connection using a close() method of MySQLConnection class.**

**Now you know how to connect to MySQL server from python let’s proceed with creating a table from Python.**

## 

## 

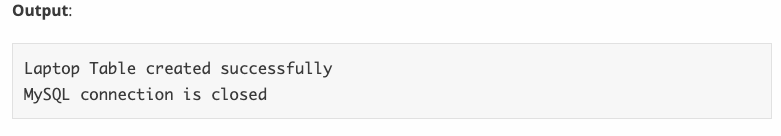
## 

## 

## **Python MySQL Create Table**

**In this section, we will learn how to create a table in MySQL from Python. In this example, I am creating a Laptop table under the Electronics database.**

****

****

**Python MySQL Connection arguments list**

As you already know we can connect to MySQL Server from python Using a mysql.connector.connect() method or the mysql.connector.MySQLConnection() class. We already discussed the four mandatory arguments required to connect the MySQL Server.

Let's see what the other connection arguments we can use to communicate with MySQL server from Python are. Following is the list of all other connection arguments and their examples.

* **port**: The TCP/IP port of the MySQL server. This value must be an integer. We can specify the different port number if MySQL server is listening to the different port. The **default value** for this port argument is ***3306.***
* **use\_unicode:** Whether to use Unicode. The **default value is True**.
* **charset:** MySQL character set to use, character set variables relate to a client’s interaction with the server. There are almost 30 to 40 charset MySQL server supports. The **default value** of the charset argument is “**utf8″.**
* **auto-commit:** Whether to auto-commit transactions. If you want to manage transactions in MySQL from Python, you need to set this value to true or false. The default value is False, i.e. the changes are not committed to the database. You need to explicitly call a commit method to persist your changes in the database.
* **get\_warnings:** To fetch warnings, this is helpful to know the connection is established but with warnings. The default value is False.
* **raise\_on\_warnings:** Whether to raise an exception on warnings. The Default value is False.
* **connection\_timeout** (connect\_timeout\*) : Timeout for the TCP and Unix socket connections. The connection terminates after this timeout expires.
* **buffered:** Whether cursor objects fetch the results immediately after executing queries. The default value is False.
* **raw:** Whether MySQL results are returned as-is, rather than converted to Python types**.** The default value is False. You can set it to true if you want a query result in python type.
* **force\_ipv6:** When setting to True, uses IPv6 when an address resolves to both IPv4 and IPv6. By default, IPv4 is used in such cases. The default value for this argument is false.
* **pool\_name**: It is the Connection pool name that you are creating or using.
* **pool\_size**: Connection pool size that you want to create. the default value is 5.
* **pool\_reset\_session**: Whether to reset session variables when the connection returned to the pool. The default is True.
* **use\_pure:** Whether to use pure Python or C Extension. If use\_pure=False then pure python module is used otherwise it connects MySQL using C extension. Moreover, if C Extension is not available, then My SQL Connector Python automatically fall back to the pure Python implementation.
* **unix\_socket**: The location of the Unix socket file. These enable communication between two processes.
* **auth\_plugin:** Authentication plugin to use. Added in 1.2.1.
* **collation:** MySQL collation to use. you can use the collation that you set while installing MySQL Server. The **default value** is **utf8\_generalW\_chiich.**
* **sql\_mode**: Set the sql\_mode session variable at connection time.

Furthermore, let's see how to use a dictionary to store all of these connection arguments.

### **Use the Dictionary to keep MySQL Connection arguments in Python**

If you have lots of connection arguments, it’s best to keep them in a dictionary and use the \*\* operator. For example, you know you require a minimum of four arguments (i.e., username, password, hostname, database name) to connect to MySQL.

Sometimes in exceptional cases, we need more than four arguments in the connect method to connect the MySQL database. Let’s understand this. For example below are three more connection arguments we can use in the connect method.

* **connection\_timeout** – Timeout for the TCP and Unix socket connections
* **Autocommit** – Whether to auto-commit transactions. The default is false
* **pool\_size** – Connection pool size if you want to use connection pooling.

You can use many other connection arguments as per your need and add them all in a dictionary and pass a dictionary to connect() method. Let’s demonstrate the above with the below example.



## 

## 

## **Change the MySQL connection timeout when connecting through Python**

Sometimes we need to change the connection timeout value if we are reading or inserting a large amount of data to the MySQL server. Then how to change the default Mysql connection timeout when connecting through python?

We can do this using a **connection\_timeout** argument of MySQL connector python to manage the timeout issues by increasing the timeout value.

connection\_timeout is the timeout value in second for the TCP and Unix socket connections. This time denotes the number of seconds the MySQL server waits to fulfill the current request. **Connection terminates If the request takes more time than this value.**

Also, you can set the following Parameters of MySQL server by executing SQL query from python to handle the connection timeout issue. Change the value of the following parameters only when the connection\_timeout argument alone can’t handle the timeout issue.

* **interactive\_timeout** – the number of seconds the server should wait for activity on an interactive connection before closing it.
* **wait\_timeout** – Number of seconds the server should wait for activity on a connection before closing it.

You can set connection\_timeout values as per your requirement. Let's see how to manage MySQL connection timeout through python using an example.

**Further reading: MySQL with Python on Ubuntu:** https://pythonspot.com/mysql-with-python/