Regular expressions, Database Programming

Module 4

Prepared by,

Rini Kurian

Assistant Professor, MCA

AJCE

Module Contents

- **Regular expressions**: Introduction, match() function, search() function, search and replace, regular expression modifiers, regular expression patterns, Character classes, special character classes, repetition cases, findall() method, compile() method.

- **Database Programming**: Connecting to a database, Creating Tables, INSERT, UPDATE, DELETE and READ operations, Transaction Control, Disconnecting from a database, Exception Handling in Databases

Regular expressions: Introduction

- ❑ A Regular Expressions (RegEx) is a special sequence of characters that uses a search pattern to find a string or set of strings.
- ❑ It can detect the presence or absence of a text by matching with a particular pattern, and also can split a pattern into one or more sub-patterns.
- ❑ Python provides a **re module** that supports the use of regex in Python.
- ❑ Its primary function is to offer a search, where it takes a regular expression and a string. Here, it either returns the first match or else none.
- ❑ The re module must be imported to use the regex functionalities in python

```
import re
```

Forming a regular expression

A regular expression can be formed by using the mix of

- Regular expression modifiers
- Regular expression patterns
- Character classes
- Special character classes
- Repetition cases

Metacharacters

Metacharacters are characters with a special meaning:

MetaCharacters	Description
\	Used to drop the special meaning of character following it
[ ]	Represent a character class
^	Matches the beginning
$	Matches the end
.	Matches any character except newline

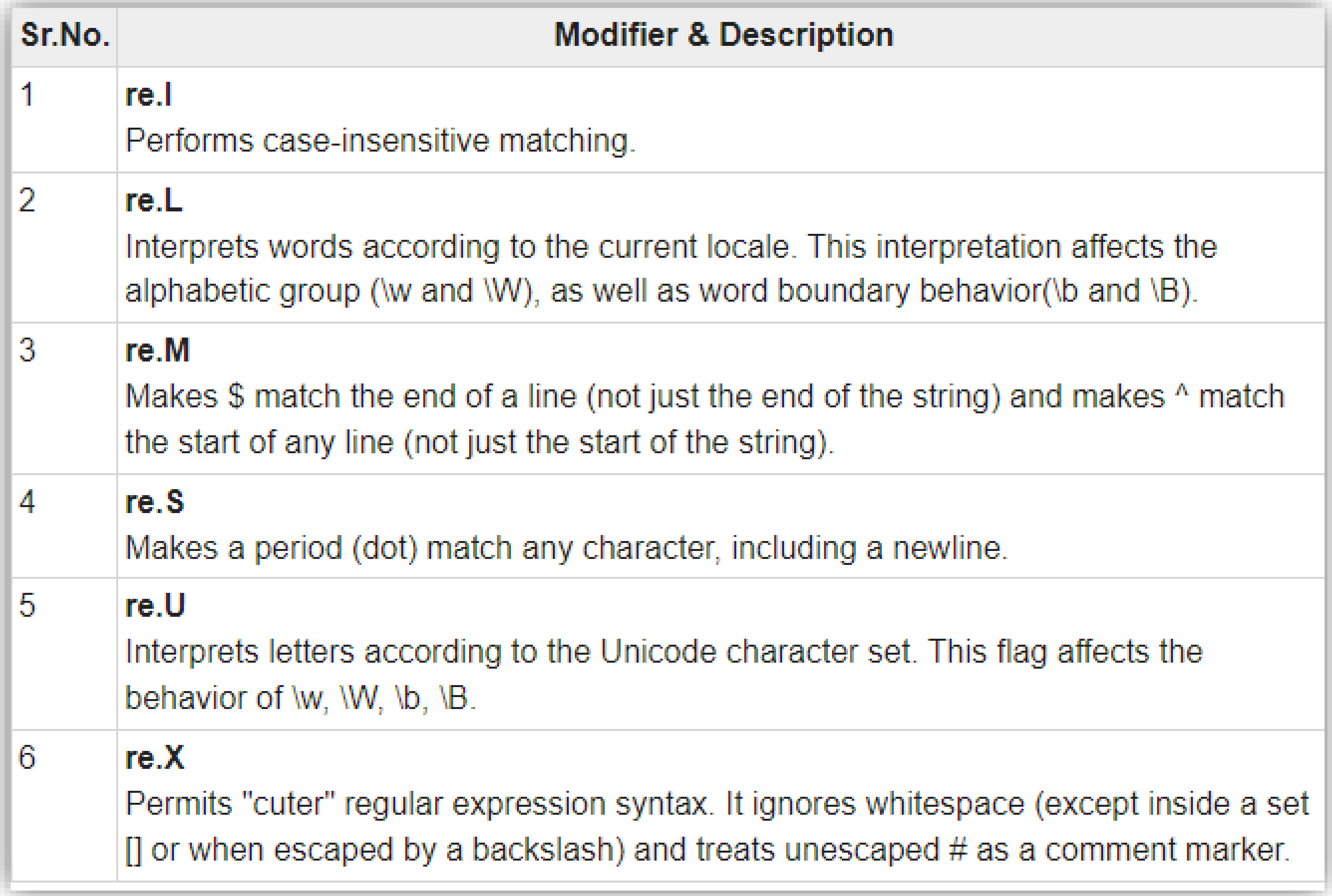
MetaCharacters	Description
|	Means OR (Matches with any of the characters separated by it.
?	Matches zero or one occurrence
*****	Any number of occurrences (including 0 occurrences)
+	One or more occurrences
{}	Indicate the number of occurrences of a preceding regex to match.
()	Enclose a group of Regex

Regular Expression Modifiers: Option Flags

Regular expression literals may include an optional modifier to control various aspects of matching.

The modifiers are specified as an optional flag.

You can provide multiple modifiers using exclusive OR (|)



Sr.No.	Modifier & Description
1	**re.I** Performs case-insensitive matching.
2	**re.L** Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior(\b and \B).
3	**re.M** Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
4	**re.S** Makes a period (dot) match any character, including a newline.
5	**re.U** Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
6	**re.X** Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker.

Regular Expression Patterns

Except for control characters, **(+ ? . * ^ $ ( ) [ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash.

Sr.No.	Pattern & Description
1	**^** Matches beginning of line.
2	**$** Matches end of line.
3	**.** Matches any single character except newline. Using m option allows it to match newline as well.
4	**[...]** Matches any single character in brackets.
5	**[^...]** Matches any single character not in brackets
6	**re*** Matches 0 or more occurrences of preceding expression.
7	**re+** Matches 1 or more occurrence of preceding expression.

Sr.No.	Pattern & Description
8	**re?** Matches 0 or 1 occurrence of preceding expression.
9	**re{ n}** Matches exactly n number of occurrences of preceding expression.
10	**re{ n,}** Matches n or more occurrences of preceding expression.
11	**re{ n, m}** Matches at least n and at most m occurrences of preceding expression.
12	**a| b** Matches either a or b.
13	**(re)** Groups regular expressions and remembers matched text.
14	**(?imx)** Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.

Sr.No.	Pattern & Description
15	**(?-imx)** Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
16	**(?: re)** Groups regular expressions without remembering matched text.
17	**(?imx: re)** Temporarily toggles on i, m, or x options within parentheses.
18	**(?-imx: re)** Temporarily toggles off i, m, or x options within parentheses.
19	**(?#...)** Comment.
20	**(?= re)** Specifies position using a pattern. Doesn't have a range.
21	**(?! re)** Specifies position using pattern negation. Doesn't have a range.

Sr.No.	Pattern & Description
22	**(?> re)** Matches independent pattern without backtracking.
23	**\w** Matches word characters.
24	**\W** Matches nonword characters.
25	**\s** Matches whitespace. Equivalent to [\t\n\r\f].
26	**\S** Matches nonwhitespace.
27	**\d** Matches digits. Equivalent to [0-9].
28	**\D** Matches nondigits.
29	**\A** Matches beginning of string.
30	**\Z** Matches end of string. If a newline exists, it matches just before newline.

Sr.No.	Pattern & Description
31	**\z** Matches end of string.
32	**\G** Matches point where last match finished.
33	**\b** Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
34	**\B** Matches nonword boundaries.
35	**\n, \t, etc.** Matches newlines, carriage returns, tabs, etc.
36	**\1...\9** Matches nth grouped subexpression.
37	**\10** Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Character class

- ✓ A "character class", or a "character set", is a **set of characters put in square brackets.**
- ✓ The regex engine matches only one out of several characters in the character class or character set.
- ✓ We place the characters we want to match between square brackets.
- ✓ A character class or set matches only a single character.
- ✓ The order of the characters inside a character class or set does not matter. The results are identical.

- ✓ We use a hyphen inside a character class to specify a range of characters.
- ✓ [0-9] matches a single digit between 0 and 9.
- ✓ Similarly for uppercase and lowercase letters we have the character class [A-Za-z]

Sr.No.	Example & Description
1	**[Pp]ython** Match "Python" or "python"
2	**rub[ye]** Match "ruby" or "rube"
3	**[aeiou]** Match any one lowercase vowel
4	**[0-9]** Match any digit; same as [0123456789]
5	**[a-z]** Match any lowercase ASCII letter
6	**[A-Z]** Match any uppercase ASCII letter
7	**[a-zA-Z0-9]** Match any of the above
8	**[^aeiou]** Match anything other than a lowercase vowel
9	**[^0-9]** Match anything other than a digit

Special character classes

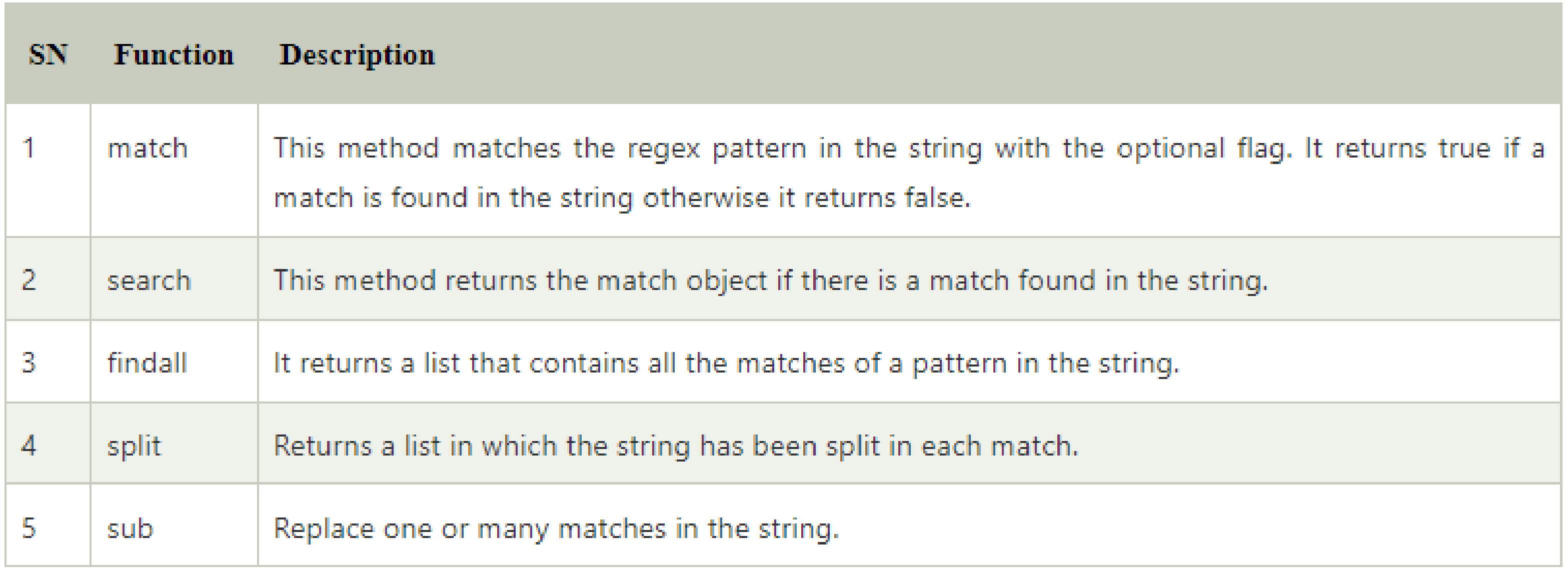
Sr.No.	Example & Description
1	**.** Match any character except newline
2	**\d** Match a digit: [0-9]
3	**\D** Match a nondigit: [^0-9]
4	**\s** Match a whitespace character: [\t\r\n\f]
5	**\S** Match nonwhitespace: [^ \t\r\n\f]
6	**\w** Match a single word character: [A-Za-z0-9_]
7	**\W** Match a nonword character: [^A-Za-z0-9_]

Repetition Cases

Sr.No.	Example & Description
1	**ruby?** Match "rub" or "ruby": the y is optional
2	**ruby*** Match "rub" plus 0 or more ys
3	**ruby+** Match "rub" plus 1 or more ys
4	**\d{3}** Match exactly 3 digits
5	**\d{3,}** Match 3 or more digits
6	**\d{3,5}** Match 3, 4, or 5 digits

Regex Functions

The following regex functions are used in the python.



SN	Function	Description
1	match	This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false.
2	search	This method returns the match object if there is a match found in the string.
3	findall	It returns a list that contains all the matches of a pattern in the string.
4	split	Returns a list in which the string has been split in each match.
5	sub	Replace one or many matches in the string.

The *match* Function

This function attempts to match RE *pattern* to *string* with optional *flags*.

Here is the syntax for this function −

```
re.match(pattern, string, flags=0)
```

Parameter & Description
pattern This is the regular expression to be matched.
string This is the string, which would be searched to match the pattern at the beginning of string.
flags You can specify different flags using bitwise OR (|).

The *re.match* function returns a **match** object on success, **None** on failure.

We use *group(num)* or *groups()* function of **match** object to get matched expression.

Match Object Method & Description
group(num=0) This method returns entire match (or specific subgroup num)
groups() This method returns all matching subgroups in a tuple (empty if there weren't any)

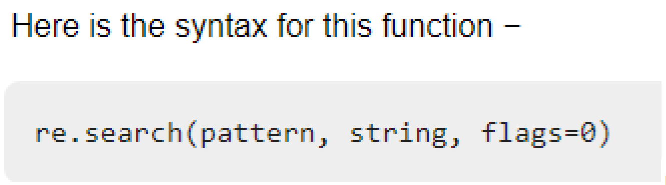
```python
#!/usr/bin/python
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

When the above code is executed, it produces following result –

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

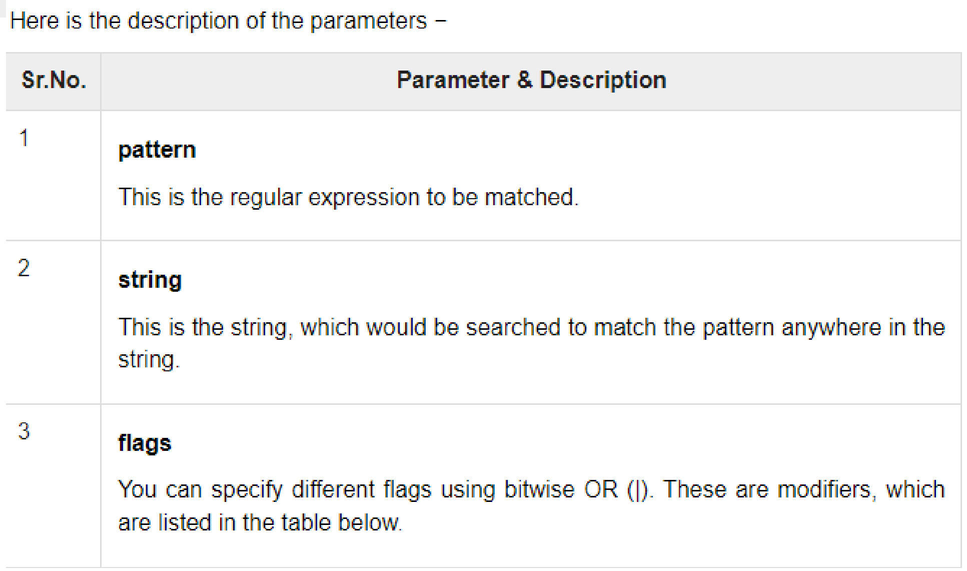
The *search* Function

This function searches for first occurrence of RE *pattern* within *string* with optional *flags*



Here is the syntax for this function –

```
re.search(pattern, string, flags=0)
```

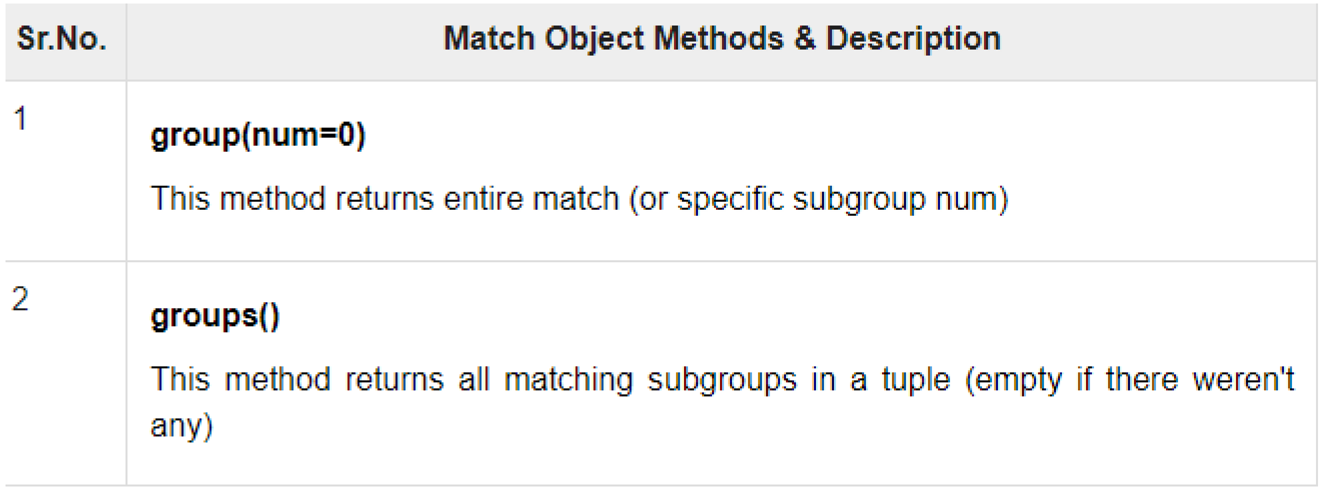


Here is the description of the parameters –

Sr.No.	Parameter & Description
1	**pattern** This is the regular expression to be matched.
2	**string** This is the string, which would be searched to match the pattern anywhere in the string.
3	**flags** You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The re.search function returns a match object on success, none on failure.

We use group(num) or groups() function of match object to get matched expression.



Sr.No.	Match Object Methods & Description
1	**group(num=0)** This method returns entire match (or specific subgroup num)
2	**groups()** This method returns all matching subgroups in a tuple (empty if there weren't any)

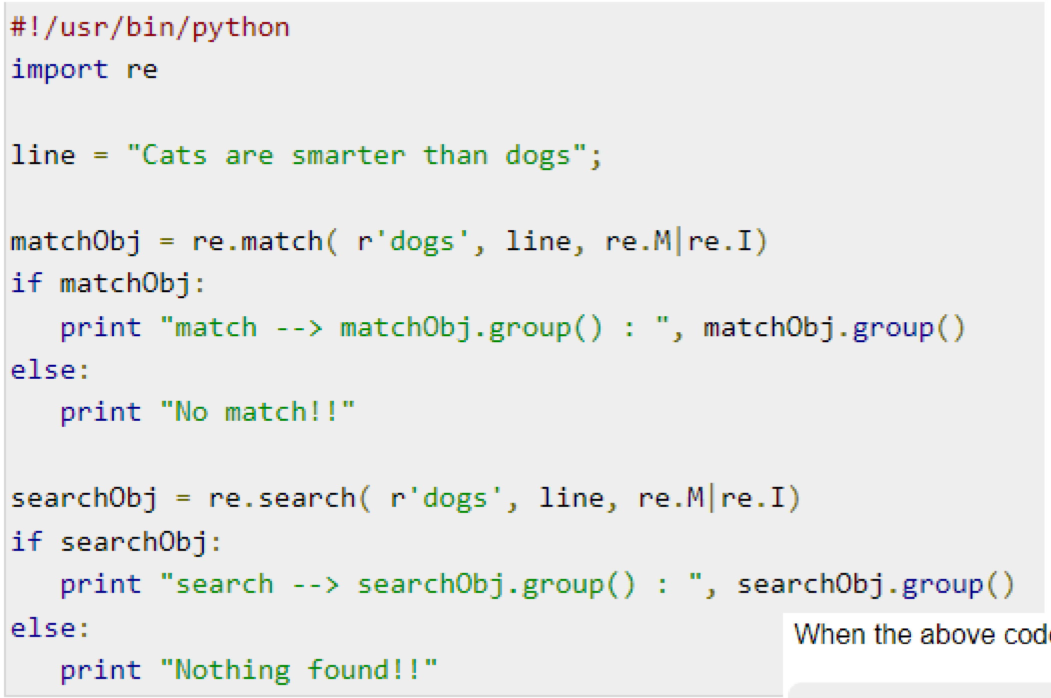
```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs";

searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)

if searchObj:
   print "searchObj.group() : ", searchObj.group()
   print "searchObj.group(1) : ", searchObj.group(1)
   print "searchObj.group(2) : ", searchObj.group(2)
else:
   print "Nothing found!!"
```

When the above code is executed, it produces following result −

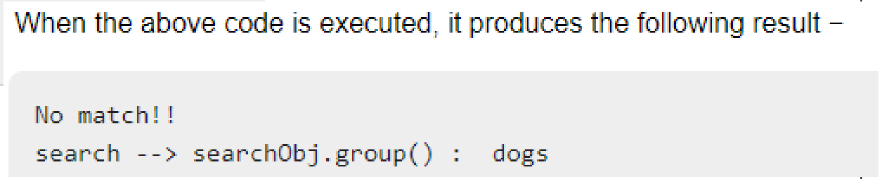
```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

Matching Versus Searching

Python offers two different primitive operations based on regular expressions: match checks for a match only at the beginning of the string, while search checks for a match anywhere in the string.



```python
#!/usr/bin/python
import re

line = "Cats are smarter than dogs";

matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
   print "match --> matchObj.group() : ", matchObj.group()
else:
   print "No match!!"

searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
   print "search --> searchObj.group() : ", searchObj.group()
else:
   print "Nothing found!!"
```

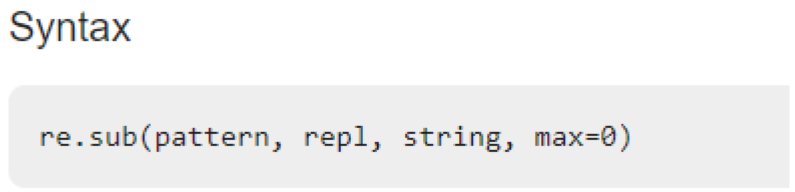


When the above code is executed, it produces the following result −

```
No match!!
search --> searchObj.group() :  dogs
```

Search and Replace(sub)

One of the most important re methods that use regular expressions is **sub**.



Syntax

```
re.sub(pattern, repl, string, max=0)
```

This method replaces all occurrences of the RE pattern in string with repl, substituting all occurrences unless max provided. This method returns modified string.

```
#!/usr/bin/python
import re

phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print "Phone Num : ", num

# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print "Phone Num : ", num
```

When the above code is executed, it produces the following result −

```
Phone Num :  2004-959-559
Phone Num :  2004959559
```

The findall Method

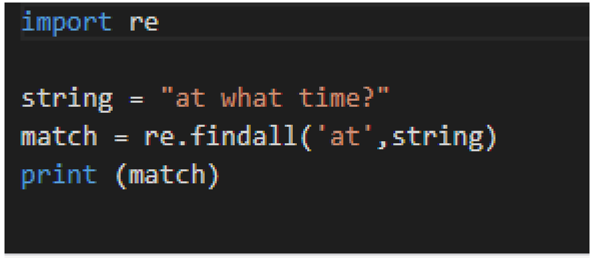
The re.findall() function returns a list of strings containing all matches of the specified pattern.

The function takes as input the following:

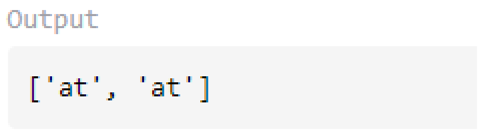
- ❖ a character pattern
- ❖ the string from which to search

Example:

The following example will return a list of all the instances of the substring at in the given string:



```
import re

string = "at what time?"
match = re.findall('at',string)
print (match)
```



Output

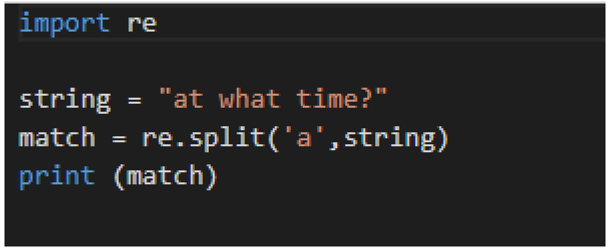
```
['at', 'at']
```

The split() function

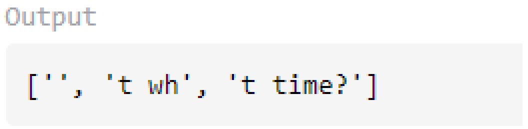
The re.split() function splits the string at every occurrence of the sub-string and returns a list of strings which have been split.

Example

Suppose we wish to split a string wherever there is an occurrence of a



```
import re

string = "at what time?"
match = re.split('a',string)
print (match)
```



Output

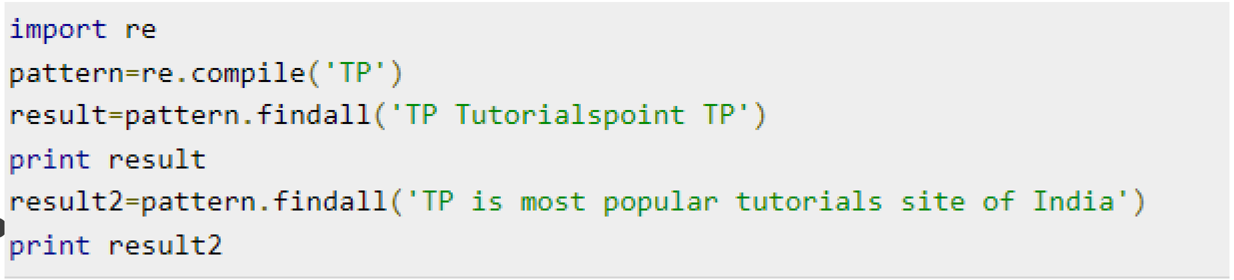
```
['', 't wh', 't time?']
```

The compile() method

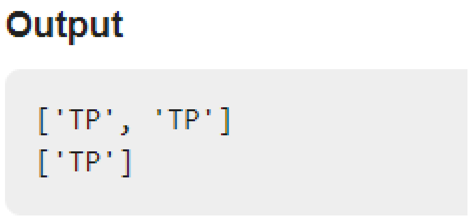
We can combine a regular expression pattern into pattern objects, which can be used for pattern matching. It also helps to search a pattern again without rewriting it.

Syntax:

re.compile(pattern, repl, string):



```
import re
pattern=re.compile('TP')
result=pattern.findall('TP Tutorialspoint TP')
print result
result2=pattern.findall('TP is most popular tutorials site of India')
print result2
```



Output

```
['TP', 'TP']
['TP']
```

Database Programming

Database

The database is a collection of organized information that can easily be used, managed, update, and they are classified according to their organizational approach

Benefits of Python Database Programming

- ➢ Programming in Python is considerably simple and efficient with compared to other languages, so as the database programming
- ➢ Python database is portable, and the program is also portable so both can give an advantage in case of portability
- ➢ Python supports SQL cursors
- ➢ It also supports Relational Database systems
- ➢ The API of Python for the database is compatible with other databases also
- ➢ It is platform-independent

Database Programming in Python

- The Python programming language has powerful features for database programming. Python supports various databases like MySQL, Oracle, Sybase, PostgreSQL, etc.
- Python also supports Data Definition Language (DDL), Data Manipulation Language (DML), and Data Query Statements.
- For database programming, the Python DB API is a widely used module that provides a database application programming interface.

DB-API (SQL-API) for Python

Python provides DB-API which is independent of any database engine and it enables you to write Python scripts to access any database engine. The Python DB-API implementation for different databases are as follows –

- ✓ For MySQL it is MySQLdb.
- ✓ For PostgreSQL it is psycopg, PyGresQL and pyPgSQL
- ✓ For Oracle it has dc_oracle2 and cx_oracle.
- ✓ For DB2 DB-API implementation is Pydb2.

You must download a separate DB API module for each database you need to access.
For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −

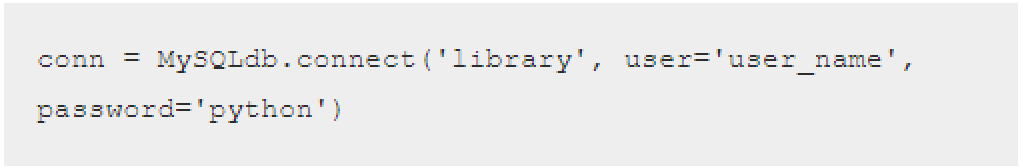
- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

Python’s DB-API consists of

- Connection Objects
- Cursor Objects
- Standard Exceptions
- Some Other Module Contents.

Connection Objects

- Connection objects in DB-API of Python create a connection with the database which is further used for different transactions.
- These connection objects are also used as representatives of the database session.
- A connection to a database in Python is created as follows:



```
conn = MySQLdb.connect('library', user='user_name',
password='python')
```

Cursor Objects

- The cursor is one of the powerful features of SQL.
- These are objects that are responsible for submitting various SQL statements to a database server.
- There are several cursor classes in **MySQLdb.cursors**:

```
cursor.execute('SELECT * FROM books')
cursor.execute('''SELECT * FROM books WHERE book_name = 'python'
AND  book_author = 'Mark Lutz')
cursor.close()
```

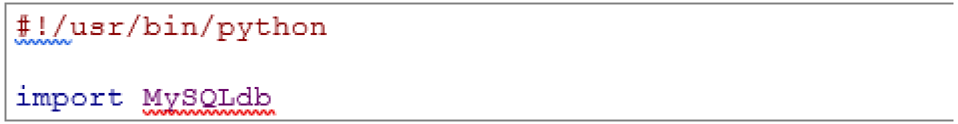
These objects represent a database cursor, which is used to manage the context of a fetch operation. Cursors created from the same connection are not isolated, i.e., any changes done to the database by a cursor are immediately visible by the other cursors. Cursors created from different connections can or can not be isolated, depending on how the transaction support is implemented.

What is MySQLdb?

MySQLdb is an interface for connecting to a MySQL database server from Python

How do I Install MySQLdb?

Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it −



```
#!/usr/bin/python

import MySQLdb
```

If it produces the following result, then it means MySQLdb module is not installed −

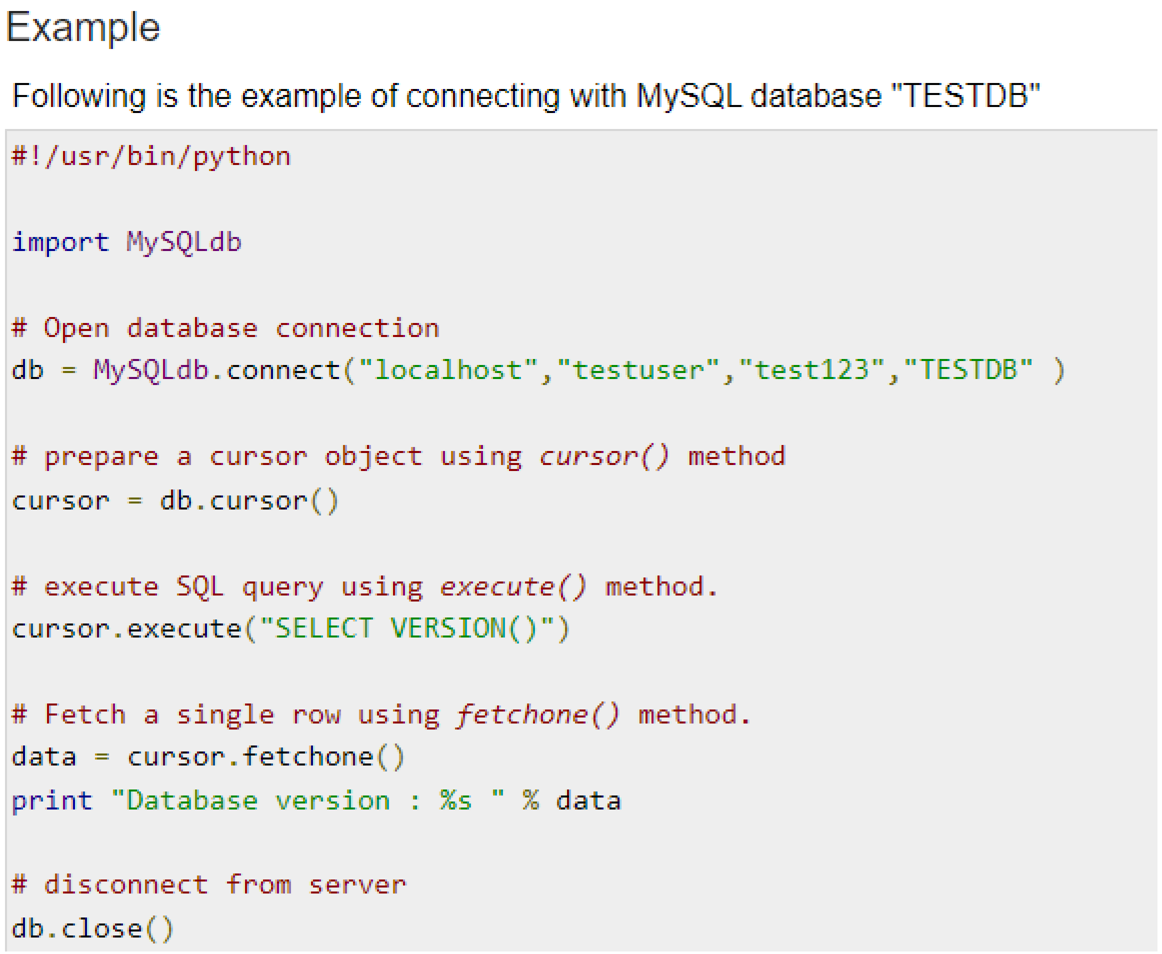
```
Traceback (most recent call last):
   File "test.py", line 3, in <module>
      import MySQLdb
ImportError: No module named MySQLdb
```

To install MySQLdb module, use the following command −

```
For Ubuntu, use the following command -

$ sudo apt-get install python-pip python-dev libmysqlclient-dev


For Python command prompt, use the following command -

pip install MySQL-python
```

Before connecting to a MySQL database, make sure of the followings −

- You have created a database TESTDB.
- You have created a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Python module MySQLdb is installed properly on your machine.



Example

Following is the example of connecting with MySQL database "TESTDB"

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print "Database version : %s " % data

# disconnect from server
db.close()
```

- If a connection is established with the datasource, then a Connection Object is returned and saved into db for further use, otherwise db is set to None.

- Next, db object is used to create a cursor object, which in turn is used to execute SQL queries.

- Finally, before coming out, it ensures that database connection is closed and resources are released.

Creating Database Table

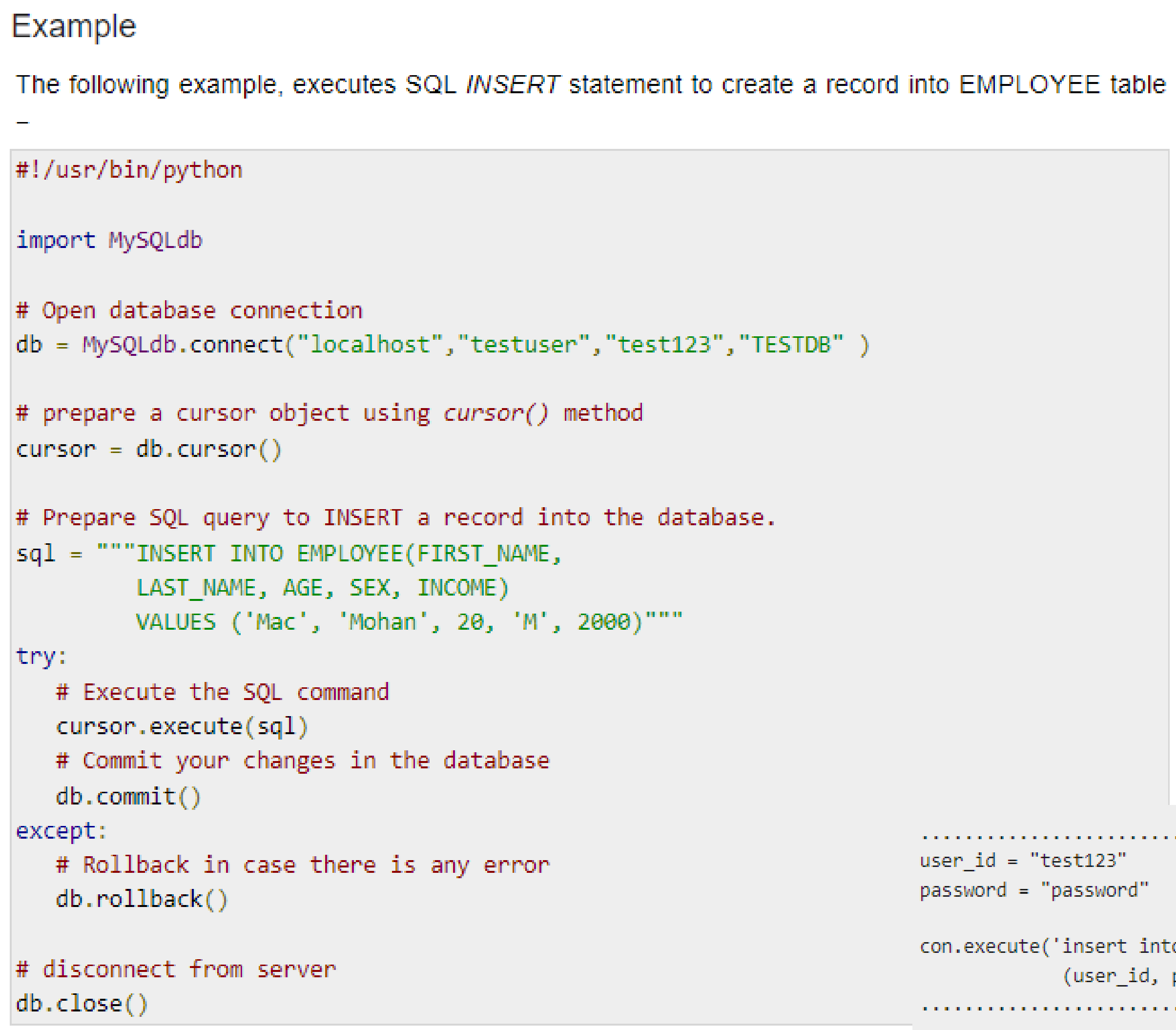
Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

Let us create Database table EMPLOYEE –

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()
```

INSERT Operation

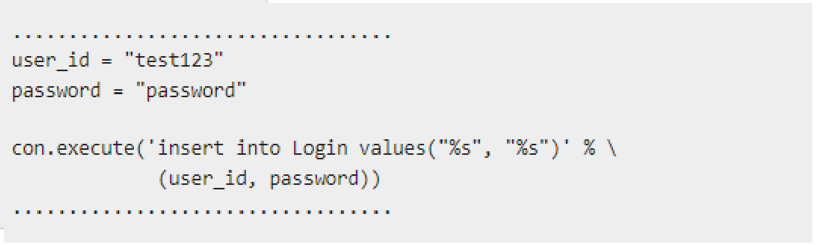
It is required when you want to create your records into a database table.



Example

The following example, executes SQL *INSERT* statement to create a record into EMPLOYEE table –

```python
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```



```python
..................................
user_id = "test123"
password = "password"

con.execute('insert into Login values("%s", "%s")' % \
             (user_id, password))
..................................
```

READ Operation

READ Operation on any database means to fetch some useful information from the database.

Once our database connection is established, you are ready to make a query into this database.

You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetech multiple values from a database table.

- ✓ **fetchone() −** It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- ✓ **fetchall() −** It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- ✓ **rowcount −** This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Example

The following procedure queries all the records from

EMPLOYEE table having salary more than 1000 −

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "SELECT * FROM EMPLOYEE \
       WHERE INCOME > '%d'" % (1000)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   for row in results:
      fname = row[0]
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]
      # Now print fetched result
      print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
             (fname, lname, age, sex, income )
except:
   print "Error: unable to fecth data"

# disconnect from server
db.close()
```

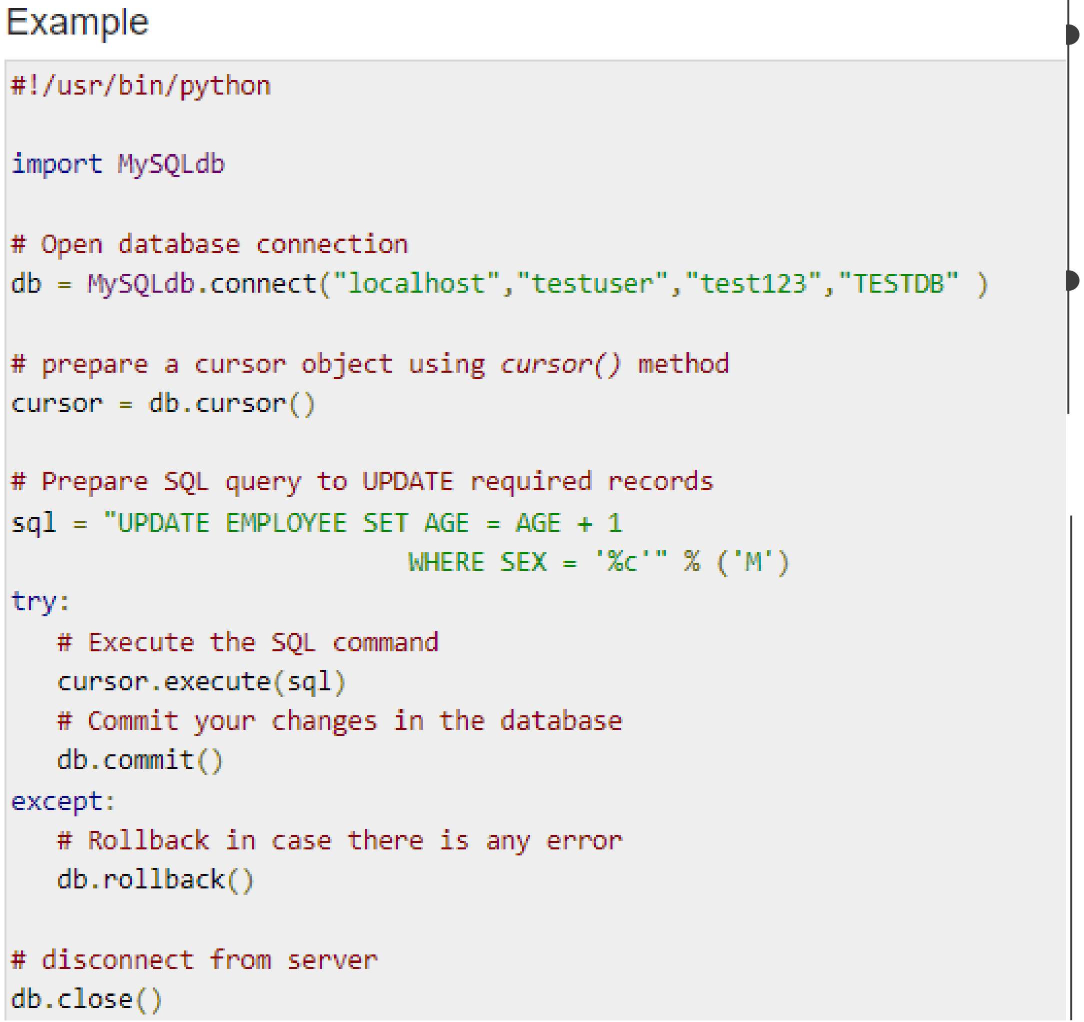
This will produce the following result −

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

Update Operation

- UPDATE Operation on any database means to update one or more records, which are already available in the database.

- The following procedure updates all the records having SEX as **'M'**. Here, we increase AGE of all the males by one year.



Example

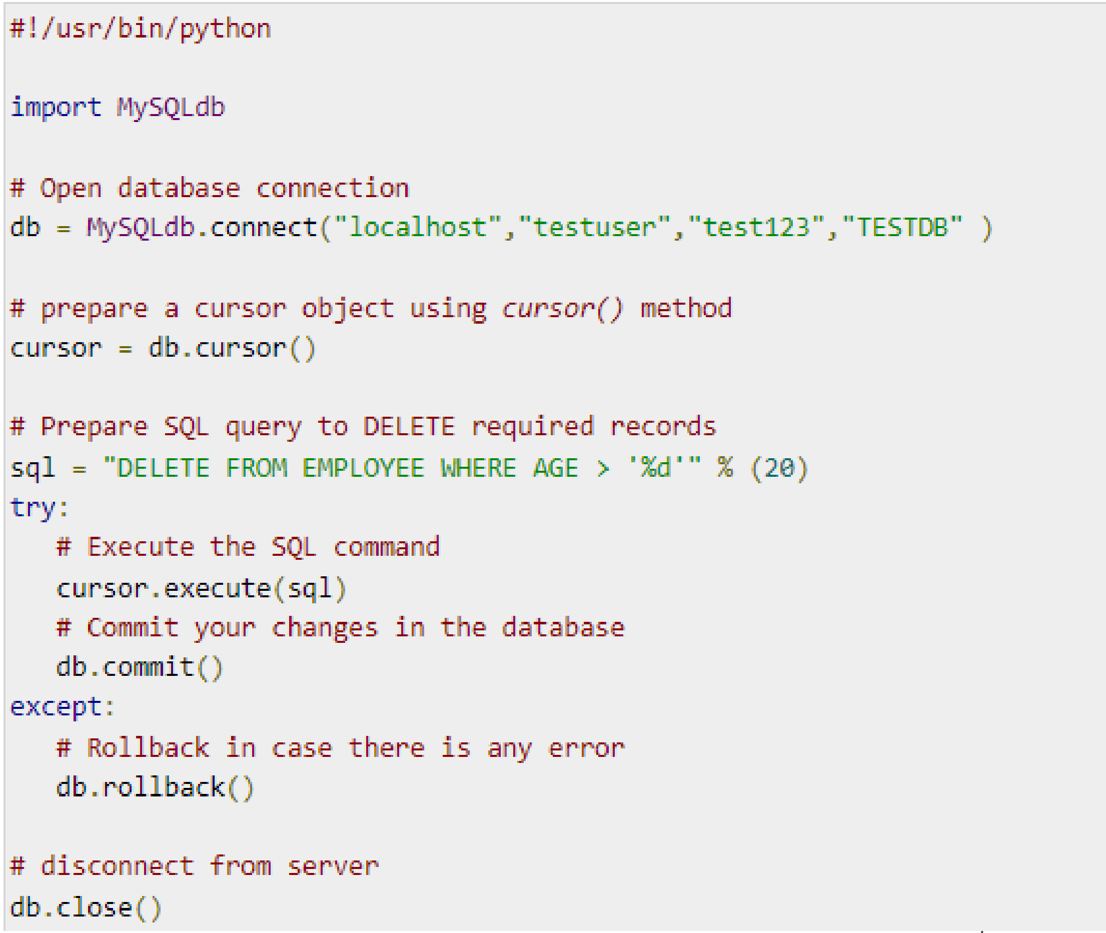
```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
                          WHERE SEX = '%c'" % ('M')
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

DELETE Operation

- DELETE operation is required when you want to delete some records from your database.

- Following is the procedure to delete all the records from EMPLOYEE where AGE is more than 20 −

Example



```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

Performing Transactions

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties −

- **Atomicity −** Either a transaction completes or nothing happens at all.
- **Consistency −** A transaction must start in a consistent state and leave the system in a consistent state.
- **Isolation −** Intermediate results of a transaction are not visible outside the current transaction.
- **Durability −** Once a transaction was committed, the effects are persistent, even after a system failure.

The Python DB API 2.0 provides two methods to either *commit* or *rollback* a transaction.

Example

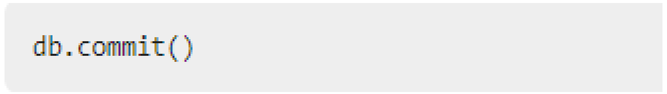
You already know how to implement transactions. Here is again similar example –

```
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()
```

COMMIT Operation

Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.

Here is a simple example to call **commit** method.



```
db.commit()
```

ROLLBACK Operation

If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use rollback() method.

Here is a simple example to call rollback() method.

```
db.rollback()
```

Disconnecting Database

To disconnect Database connection, use close() method.

```
db.close()
```

If the connection to a database is closed by the user with the close() method, any outstanding transactions are rolled back by the DB. However, instead of depending on any of DB lower level implementation details, your application would be better off calling commit or rollback explicitly.

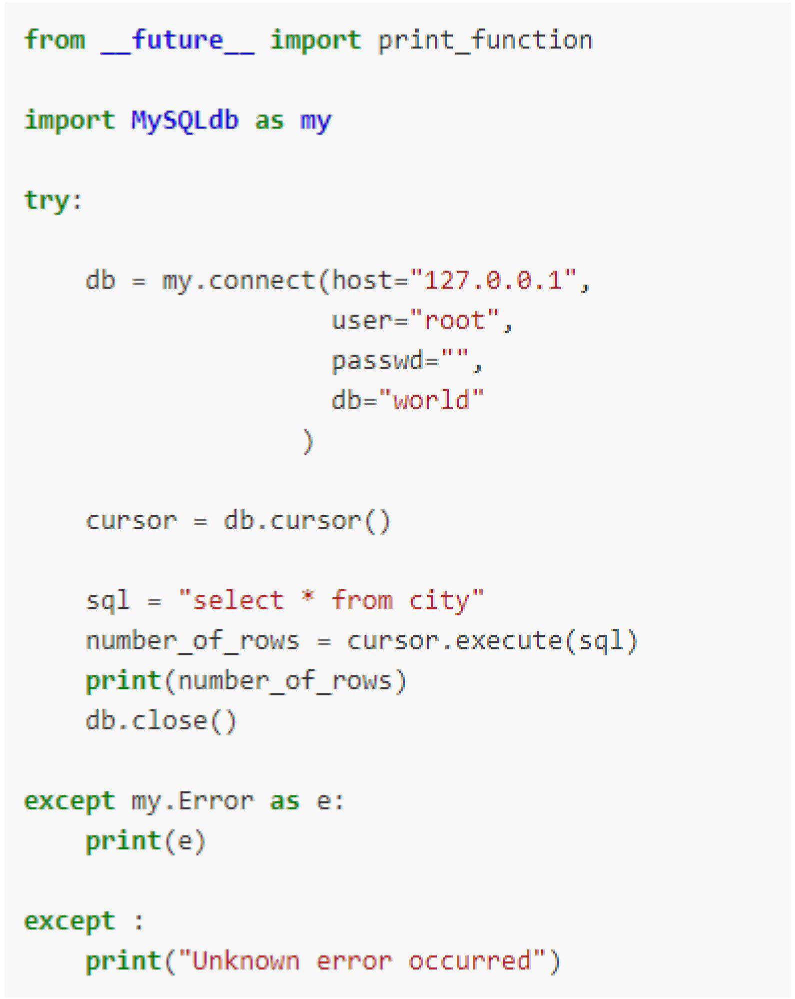
Handling Errors

There are many sources of errors. A few examples are a syntax error in an executed SQL statement, a connection failure, or calling the fetch method for an already canceled or finished statement handle.

The DB API defines a number of errors that must exist in each database module. Your Python scripts should handle these errors, but before using any of the above exceptions, make sure your MySQLdb has support for that exception.

Sr.No.	Exception & Description
1	**Warning** Used for non-fatal issues. Must subclass StandardError.
2	**Error** Base class for errors. Must subclass StandardError.
3	**InterfaceError** Used for errors in the database module, not the database itself. Must subclass Error.
4	**DatabaseError** Used for errors in the database. Must subclass Error.

Sr.No.	Exception & Description
5	**DataError** Subclass of DatabaseError that refers to errors in the data.
6	**OperationalError** Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter.
7	**IntegrityError** Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys.
8	**InternalError** Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active.
9	**ProgrammingError** Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you.
10	**NotSupportedError** Subclass of DatabaseError that refers to trying to call unsupported functionality.



```
from __future__ import print_function

import MySQLdb as my

try:

    db = my.connect(host="127.0.0.1",
                    user="root",
                    passwd="",
                    db="world"
                   )

    cursor = db.cursor()

    sql = "select * from city"
    number_of_rows = cursor.execute(sql)
    print(number_of_rows)
    db.close()

except my.Error as e:
    print(e)

except :
    print("Unknown error occurred")
```

MySQLdb has MySQLdb.Error exception, a top level exception that can be used to catch all database exception raised by MySQLdb module.

Thankyou !!!