**Quickstart usage of various features:**

import \_mssql  
conn = \_mssql.connect(server='SQL01', user='user', password='password', \  
    database='mydatabase')  
conn.execute\_non\_query('CREATE TABLE persons(id INT, name VARCHAR(100))')  
conn.execute\_non\_query("INSERT INTO persons VALUES(1, 'John Doe')")  
conn.execute\_non\_query("INSERT INTO persons VALUES(2, 'Jane Doe')")

# how to fetch rows from a table  
conn.execute\_query('SELECT \* FROM persons WHERE salesrep=%s', 'John Doe')  
for row in conn:  
    print "ID=%d, Name=%s" % (row['id'], row['name'])

# examples of other query functions  
numemployees = conn.execute\_scalar("SELECT COUNT(\*) FROM employees")  
numemployees = conn.execute\_scalar("SELECT COUNT(\*) FROM employees WHERE name LIKE 'J%'")    # note that '%' is not a special character here  
employeedata = conn.execute\_row("SELECT \* FROM employees WHERE id=%d", 13)

# how to fetch rows from a stored procedure  
conn.execute\_query('sp\_spaceused')   # sp\_spaceused without arguments returns 2 result sets  
res1 = [ row for row in conn ]       # 1st result  
res2 = [ row for row in conn ]       # 2nd result

# how to get an output parameter from a stored procedure  
sqlcmd = """  
DECLARE @res INT  
EXEC usp\_mystoredproc @res OUT  
SELECT @res  
"""  
res = conn.execute\_scalar(sqlcmd)

# how to get more output parameters from a stored procedure  
sqlcmd = """  
DECLARE @res1 INT, @res2 TEXT, @res3 DATETIME  
EXEC usp\_getEmpData %d, %s, @res1 OUT, @res2 OUT, @res3 OUT  
SELECT @res1, @res2, @res3  
"""  
res = conn.execute\_row(sqlcmd, (13, 'John Doe'))

# examples of queries with parameters  
conn.execute\_query('SELECT \* FROM empl WHERE id=%d', 13)  
conn.execute\_query('SELECT \* FROM empl WHERE name=%s', 'John Doe')  
conn.execute\_query('SELECT \* FROM empl WHERE id IN (%s)', ((5, 6),))  
conn.execute\_query('SELECT \* FROM empl WHERE name LIKE %s', 'J%')  
conn.execute\_query('SELECT \* FROM empl WHERE name=%(name)s AND city=%(city)s', \  
    { 'name': 'John Doe', 'city': 'Nowhere' } )  
conn.execute\_query('SELECT \* FROM cust WHERE salesrep=%s AND id IN (%s)', \  
    ('John Doe', (1, 2, 3)))  
conn.execute\_query('SELECT \* FROM empl WHERE id IN (%s)', (tuple(xrange(4)),))  
conn.execute\_query('SELECT \* FROM empl WHERE id IN (%s)', \  
    (tuple([3, 5, 7, 11]),))

conn.close()

Please note the usage of iterators and ability to access results by column name. Also please note that parameters to connect method have different names than in pymssql module.

**An example of exception handling:**

import \_mssql  
try:  
    conn = \_mssql.connect(server='SQL01', user='user', password='password', \  
        database='mydatabase')  
    conn.execute\_non\_query('CREATE TABLE t1(id INT, name VARCHAR(50))')  
except \_mssql.MssqlDatabaseException,e:  
    if e.number == 2714 and e.severity == 16:  
        # table already existed, so quieten the error  
    else:  
        raise # re-raise real error  
finally:  
    conn.close()

### pymssql examples (strict DB-API compliance):

import pymssql  
conn = pymssql.connect(host='SQL01', user='user', password='password', database='mydatabase')  
cur = conn.cursor()  
cur.execute('CREATE TABLE persons(id INT, name VARCHAR(100))')  
cur.executemany("INSERT INTO persons VALUES(%d, %s)", \  
    [ (1, 'John Doe'), (2, 'Jane Doe') ])  
conn.commit()  # you must call commit() to persist your data if you don't set autocommit to True  
  
cur.execute('SELECT \* FROM persons WHERE salesrep=%s', 'John Doe')  
row = cur.fetchone()  
while row:  
    print "ID=%d, Name=%s" % (row[0], row[1])  
    row = cur.fetchone()  
  
# if you call execute() with one argument, you can use % sign as usual  
# (it loses its special meaning).  
cur.execute("SELECT \* FROM persons WHERE salesrep LIKE 'J%'")  
  
conn.close()  
  
# You can also use iterators instead of while loop. Iterators are DB-API extensions, and are available since pymssql 1.0.

### Rows as dictionaries

Since pymssql 1.0.2 rows can be fetched as dictionaries instead of tuples. This allows for accessing columns by name instead of index. Note the as\_dict argument.

import pymssql  
conn = pymssql.connect(host='SQL01', user='user', password='password', database='mydatabase', as\_dict=True)  
cur = conn.cursor()  
  
cur.execute('SELECT \* FROM persons WHERE salesrep=%s', 'John Doe')  
for row in cur:  
    print "ID=%d, Name=%s" % (row['id'], row['name'])  
  
conn.close()

### Calling stored procedures

As of pymssql 2.0.0 stored procedures can be called using the rpc interface of db-lib.

import pymssql  
conn = pymssql.connect(host='SQL01', user='user', password='password', database='mydatabase', as\_dict=True)  
cur = conn.cursor()  
  
cur.callproc('findPerson', ('John Doe',))  
for row in cur:  
    print "ID=%d, Name=%s" % (row['id'], row['name'])  
  
conn.close()