```
#!/usr/bin/python
################################################################################
# Example PostgreSQL Access From Python
# =====================================
# There are actually two APIs for accessing PostgreSQL from Python. The first
# is the newer Python Database API (DB-API) which is an RDMBS independent API
# for Python. The second is the older PostgreSQL specific "pg" module API that
# ships as part of the PostgreSQL distribution. The simpler of the two
# interfaces is the older pg module, however it is less portable than the
# DB-API and it more likely to be deprecated in the future.
#
# This example code walks through a simple set of database operations using
# both the available PostgreSQL APIs. Note that in real world code you would
# expect a lot more error handling code to deal with the various exceptions
# each database call can raise.
#
################################################################################
# Global Variables - These are used by both API examples.

# Host Details
host = 'postgres.inf.ed.ac.uk'
port = 5432

# You will need to change these to your specific connection details.
user = 'example'
dbname = 'example'
passwd = 'foobar'

################################################################################
# Python DB-API Example
#
# The Python DB-API allows the python code below to be used on any DB-API
# supported database. The specific SQL strings may have to be modified for
# specific databases though.

import pgdb

def pgdbExample():
    """See: http://www.python.org/peps/pep-0249.html"""

    # DB-API needs a data source name in this format.
    dsn = host + ':' + dbname

    # By default a transaction is implicitly started when the connection is
    # created in DB-API. Unfortunately it is not possible to stop this in the
    # pgdb module, other DB-API implementations do allow different behaviour.
    connection = pgdb.connect(dsn=dsn, user=user, password=passwd)

    # A cursor object (not necessarily a PostgreSQL cursor - depends on the
    # module implementation) is required for all database operations.
    cursor = connection.cursor()

    # Create an example table.
    cursor.execute("CREATE TABLE test(code SERIAL PRIMARY KEY, data TEXT);")

    # Put something in the table. The "code" attribute is populated but
    # the implicit trigger on the SERIAL type. When inserting many rows into
    # a table the executemany() method is a better option.
    cursor.execute("INSERT INTO test (data) VALUES ('Hello World!');")

    # Commit what we've done so far. This implicitly starts a new transaction
    # within the connection object.
    connection.commit()

    # Fetch all entries in our example table.
    cursor.execute("SELECT * FROM test;")
```

```
    print "There were " + str(cursor.rowcount) + " rows in the table:\n"

    # Pull the data into python data structures. Note that this is not the
    # best way to deal with very large sets of data, the fetchone() or
    # fetchmany() method would be better. This gives the python module the
    # option of using PostgreSQL cursors or similar efficiency tricks.
    tuples = cursor.fetchall()

    # Print out table data.
    for (code, data) in tuples:
        print "Code: " + str(code) + ", Data: " + str(data)

    # Clean up.
    cursor.execute("DROP TABLE test;")
    connection.commit()

    # Close database connection.
    cursor.close()
    connection.close()


##############################################################################
# PostgreSQL Specific pg Module Example
#
# The older "pg" module API allows lower level access to the PostgreSQL
# database and metadata but is restricted to only being usable with PostgreSQL
# databases.

import pg

def pgExample():
    """See: http://www.postgresql.org/docs/7.3/static/pygresql.html"""

    # Open a connection to the database using the pg module's DB class.
    db = pg.DB(dbname=dbname, host=host, user=user, passwd=passwd)

    # Start a transaction.
    db.query("BEGIN;")

    # Create and example table.
    db.query("CREATE TABLE test(code SERIAL PRIMARY KEY, data TEXT);")

    # The DB class in the pg module has a nice method that allows us to
    # pass it a dictionary and have it insert it into the database.
    db.insert('test', {'data': 'Hello World!'})

    # Commit the transaction.
    db.query('END;')

    # Start another transaction.
    db.query("BEGIN;")

    # Fetch all entries from out example table.
    result = db.query("SELECT * FROM test;")

    print "There were " + str(result.ntuples()) + " rows in the table:\n"

    # Pull the data into python data structures.
    tuples = result.getresult()

    # Print out table data.
    for (code, data) in tuples:
        print "Code: " + str(code) + ", Data: " + str(data)

    # Clean up.
    db.query("DROP TABLE test;")
    db.query('END;')
```

```
    db.close()

################################################################################

if __name__ == "__main__":
    print "This is not a command line module. Import it into your code."

################################################################################
```