**PostgreSQL and Python Lab**

import psycopg2

# Connect to an existing database

conn = psycopg2.connect(host="something", user="user", password="pass", dbname="db")

# Open a cursor to perform database operations

cur = conn.cursor()

# Execute a command: this creates a new table

cur.execute("CREATE TABLE test (id serial PRIMARY KEY, num integer, data varchar);")

# Pass data to fill a query placeholders and let psycopg2 perform

# the correct conversion (no more SQL injections!)

cur.execute("INSERT INTO test (num, data) VALUES (%s, %s)", (100, "abcdef"))

# Query the database and obtain data as Python objects

cur.execute("SELECT \* FROM test;")

cur.fetchone() # returns (1, 100, "abcdef")  
  
# Or get multiple results and iterate through them:

cur.execute("SELECT \* FROM test;")

rows = cur.fetchall() # fetch all the resulting rows at once

for row in rows:

print(row) # row is a tuple like (1, 100, “abcdef”)

# Or can iterate over the cursor directly:

cur.execute("SELECT \* FROM test;")

for row in cur:

print(row)

# Make the changes to the database persistent

conn.commit()

# Close communication with the database

cur.close()

conn.close()  
  
**Instructions:**

1. Clone the repl link from the class webpage to your own repl.
   * Click the green "run" button at the top to get started (we won't need to do this again).
   * Edit the code in main.py to use your username/password. (They are both your Rhodes username).
   * Once you see "Seems ok" you're good to go!
   * Click "Shell"
   * You can now run python programs manually by typing "python *filename*.py"
2. Examine how the initial tables are created.
   * Open schema.sql and look at the contents.
   * Run create-tables.py to create empty tables (you will need to use your own username/password).
3. Examine how the users table is populated.
   * Open test-users.py and look through the contents.
   * Examine dump\_users().
   * Examine add\_user().
   * Call add\_user a few times from main() to add a user manually.
   * Use print/input statements to let the user add users to the database by being prompted.
   * Examine add\_users\_from\_csv() and call from main() to populate the users table with the CSV data.
4. Duplicate the functionality of test-users.py in test-classes.py, through importing from CSV.
   * You will need to add a schema to schema.sql and re-run create-tables.py to create the empty classes table. See below for the schemas.
5. Add a new table to let students enroll in classes. Edit schema.sql to do this. Rerun create-tables.py to make the new table. (You will have to re-run test-users.py and test-classes.py to re-add the content to the tables.)
6. Write a new Python program to let the user (from the keyboard) choose a user and a class and add the user into the class. Display a list of users and a list of classes and let the user pick which class they want. Make sure they can't add a class at the same time as a class they already have. If time, make it so the list of classes displayed won't show classes that conflict in time with their current schedule.

**Final database schema**

drop table if exists users;

create table users (

userid serial primary key not null,

firstname varchar(80) not null,

lastname varchar(80) not null

);

drop table if exists classes;

create table classes (

classid serial primary key not null,

classnumber varchar(10) not null,

classname varchar(80) not null,

classtime int not null,

profname varchar(80) not null

);

drop table if exists enroll;

create table enroll (

userid int not null,

classid int not null

);