>>

Python (ii)

^ >>

❖ Python + Psycopg2 (recap)

psycopg2 is a Python module providing access to PostgreSQL DBs

Standard usage:

```
import psycopg2   # include the module definitions
try:
   connnection = psycopg2.connect("dbname=Datatase")
   cursor = connnection.cursor()
   cursor.excute("SQL Query")
   for tuple in cursor.fetchall():
      # do something with next tuple
   cursor.close()
   connection.close()
except:
   print("Database error")
```

These slides aim to give more details on how Pyscopg2 used in practice

<< ^ >>

❖ Python + Psycopg2 (recap) (cont)

`connection`

- handle giving authenticated access for a given user on a given DB
- provides creation of `cursor`s to interact with database

`cursor`

- pipeline between a Python program and a PostgreSQL DB
- send SQL statements down pipeline as strings
- read results up pipeline as Python (list of) tuples

<< ^ >>

❖ Python + Psycopg2 (recap) (cont)

Python vs PostgreSQL data types ...

Strings:

- in Python: written with "..." or '...', including `\x`
- converted to SQL strings e.g. "O'Reilly" → 'O''Reilly'
- Python supports """....""" multi-line strings (useful for SQL queries)

Tuples:

- n Python: contain multiple hetergenous values (cf. C `struct`)
- similar to PostgreSQL composite (tuple) types
- written as: (val_1, val_2, ..., val_n) (note that (val_1) is not a tuple)
- examples: (1, 2, 3), (1, "John", 3.14), (1,),

❖ Examples

Example database: beers2

```
Beers( id:int, name:text, brewer:int )

Brewers( id:int, name:text, country:text )

Bars( id:int, name:text, addr:text, license:int )

Drinkers( id:int, name:text, addr:text, phone:text )

Likes( drinker:int, beer:int )

Sells( bar:int, beer:int, price:float )

Frequents( drinker:int, bar:int )
```

<< ^ >>

❖ Examples (cont)

Assume that the following code samples are wrapped in

```
import sys
import psycopg2
conn = None
try:
    conn = psycopg2.connect("dbname=beers2")
    ... example code ...
except psycopg2.Error as err:
    print("database error:",err)
finally:
    if (conn):
        conn.close()
    print("finished with database")
```

<< ^ >>

❖ Examples (cont)

Example: a list of brewers and their countries as `brewers.py`

```
cur = conn.cursor()
cur.execute("""
select name, country from Brewers order by name
""")
for tuple in cur.fetchall():
   name, country = tuple
   print(name + ", " + country)
```

```
$ python3 brewers.py
Brew Dog, Scotland
Bridge Road Brewers, Australia
Caledonian, Scotland
Carlton, Australia
Cascade, Australia
...
```

<< ^ >>

❖ Examples (cont)

Example: a list of brewers and their countries as bfrom.py

```
cur = conn.cursor()
qry = "select name from Brewers where country = %s"
country = sys.argv[1]
cur.execute(qry, [country])
for tuple in cur.fetchall():
   print(tuple[0])

$ python3 bfrom.py Scotland
Caledonian
Brew Dog
```

❖ Examples (cont)

Example: print beers preceded by the brewer as `beers.py`

```
cur = conn.cursor()
qry = """
select b.name, r.name
from   Brewers r join Beers b on (b.brewer=r.id)
"""
cur.execute(qry)
for tuple in cur.fetchall():
   print(tuple[1] + " " + tuple[0])

$ python3 beers.py
Caledonian 80/-
James Squire Amber Ale
Sierra Nevada Bigfoot Barley Wine
...
```

<< ^ >>

❖ Examples (cont)

Example: most expensive beer as `expensive.py`

```
cur = conn.cursor()
qry = """
select b.name, s.price
from   Beers b join Sells s on (b.id = s.beer)
where  s.price = (select max(price) from Sells)
"""
cur.execute(qry)
for tuple in cur.fetchall():
   print(tuple[0] + " @ " + str(tuple[1]))

$ python3 beers.py
Sink the Bismarck @ 25.0
```

<< ^ >>

❖ Examples (cont)

Example: list beers, bar+price where sold, average price as `beers1.py`

```
$ python3 beers1.py
...
New
        Australia Hotel @ 3.0
        Coogee Bay Hotel @ 2.25
        Lord Nelson @ 3.0
        Marble Bar @ 2.8
        Regent Hotel @ 2.2
        Royal Hotel @ 2.3
        Average @ 2.591666666666667
Nirvana Pale Ale
        Not sold anywhere
Old
        Coogee Bay Hotel @ 2.5
        Marble Bar @ 2.9
        Royal Hotel @ 2.65
        Average @ 2.6833333333333336
Old Admiral
        Lord Nelson @ 3.75
        Average @ 3.75
...
```

<< ^ >>

❖ Examples (cont)

```
cur = conn.cursor()
qry = "select id, name from Beers"
cur.execute(qry)
for tuple in cur.fetchall():
    q2 = """select b.name, s.price
         from Bars b join Sells s on (b.id=s.bar)
         where s.beer = %s"""
    print(tuple[1])
    cur.execute(q2, [tuple[0]])
    n, tot = 0, 0.0
    for t in cur.fetchall():
        print("\t"+t[0],"@",t[1])
        n = n + 1
        tot = tot + t[1]
    if n > 0:
        print("\tAverage @", tot/n)
    else:
        print("\tNot sold anywhere")
```

<< ^ >>

❖ Poor Usage of Python+SQL

Should generally avoid

```
cur.execute("select x,y from R")
for tup in cur.fetchall():
   q = "select * from S where id=%s"
   cur.execute(q, [tup[0]])
   for t in cur.fetchall():
      ... process t ...
```

More efficiently done as e.g.

```
qry = """
select *
from   R join S on (R.x = S.id)
"""
for tup in cur.fetchall():
   ... process tup ...
```

<< ^ >>

❖ Calling PostgreSQL functions

Two ways to call PostgreSQL functions

```
# using a standard function call from SQL
cur.execute("select * from brewer(5)")
t = cur.fetchone()
print(t[0])

# using special callproc() method
# parameters supplied as a list of values/vars
cur.callproc("brewer",[5])
t = cur.fetchone()
print(t[0])
```

`brewer(int) returns text` returns a brewer's name, given their id

<< ^

❖ Other Psycopg2 Tricks

`cur.execute(SQL Statement)`

- clearly the SQL statement can be `SELECT`
- can also be `UPDATE` or `DELETE`
- can also be a meta-data statement, e.g.
  - `CREATE TABLE`, `DROP TABLE`, `CREATE VIEW`, ...

`cur.fetchmany(#tuples)`

- gets a list of the next `#tuples` tuples
- could replace PLpgSQL `LIMIT` in some contexts

For many more examples, see Psycopg2 documentation and tutorials

Produced: 25 Mar 2021