Accessing databases from Python

Dr. Andrew C.R. Martin
andrew.martin@ucl.ac.uk
http://www.bioinf.org.uk/

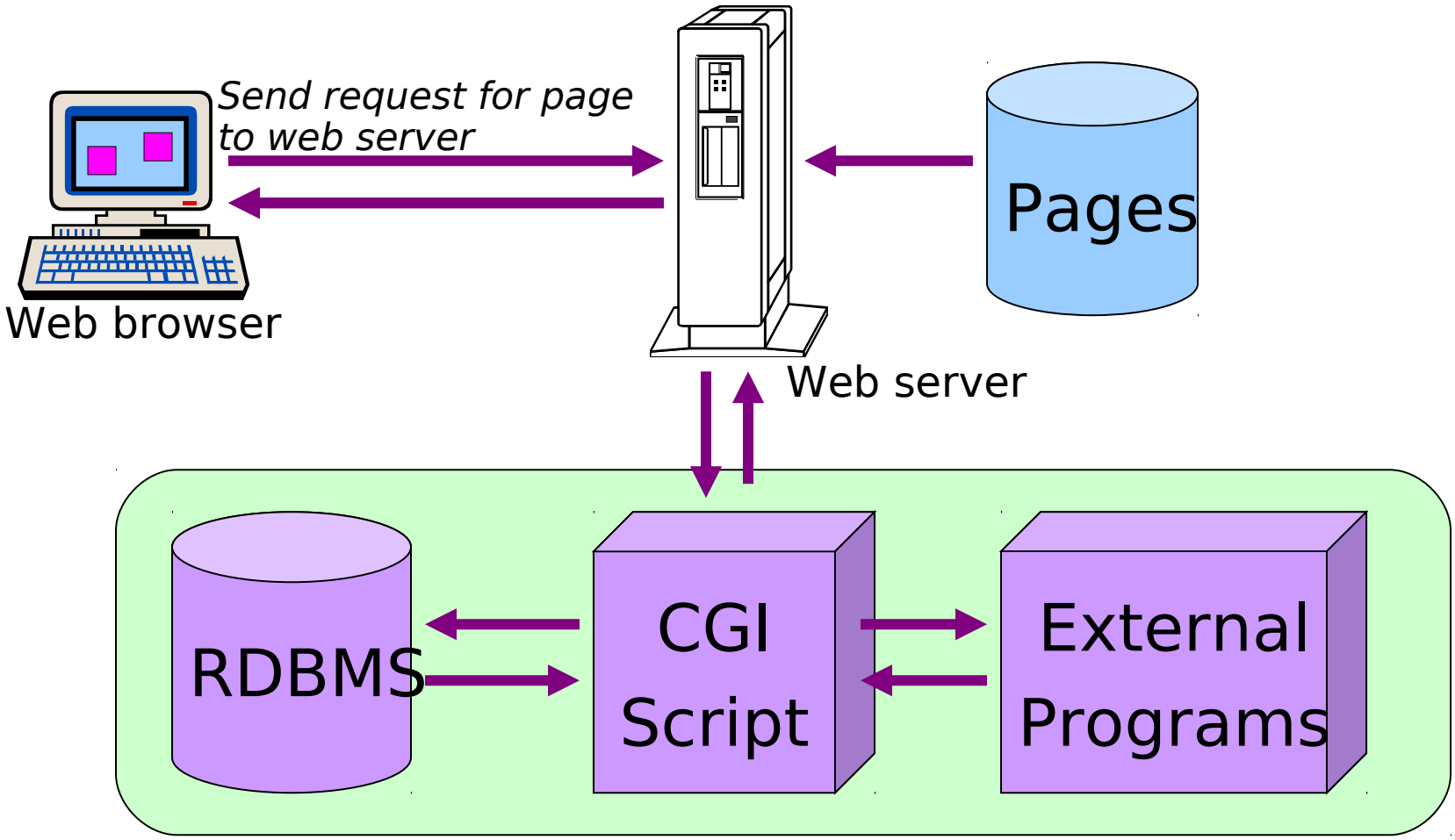
Aims and objectives

- Understand **the need to access databases from Python**
- Database **APIs**
- **PEP249:** Python Database API Specification V2.0
- Be able to **write a Python script** to read from or write to a database
- **PRACTICAL**: write a script to read from a database

UCL

Why access a database from a scripting language?

Why access a database from Python?



CGI can extract parameters sent with the page request

Why access a database from Python?

Populating databases

- Need to pre-process and re-format data into SQL
- Intermediate storage during data processing

Reading databases

- Good database design can lead to complex queries (wrappers)
- Need to extract data to process it

Database APIs

Why a standardized API?

- Many relational databases available.
- **Commercial** examples:
  - Oracle
  - DB/2
  - SQLServer
  - Sybase
  - Informix
  - Interbase
- **Open source** examples:
  - PostgreSQL
  - mySQL

A standardized API allows us to access all these databases in the same way

Why a standardized API?

- Databases use a **common query language**:
  - **‘structured query language’** (SQL)

- Queries can easily be ported between different database software
  - Minor variations in more advanced features
  - Proprietary extensions

The easy way?

- Can call command-line interface from within your program.
  - (We will return to subprocesses)

```
import subprocess

exe = 'mysql --batch --silent -e "SELECT * FROM test" test'
result = subprocess.check_output(exe, shell=True)
retval = str(retval, 'utf-8')
tuples = result.split("\n")

for tuple in tuples:
    fields = tuple.split("\t")
    for field in fields:
        print (field, "   ", end="")
    print ("")
```

Inefficient: new process for each database access

Subprocesses: when you don't need the output...

```
import os
os.system(“cmd args”)
```

e.g.

```
os.system(“mkdir /tmp/mytmpdir”)
```

You can access output through redirection and file reading

```
os.system(“cmd args > tmpfile”)
```

e.g.

```
os.system(“ls > /tmp/ls.txt”)
```

When you need the output...

```
import os
stream = os.popen("cmd args")
```

✔ As os.system() but stream is a file handle that can be used in the usual way

---

```
import subprocess
retval=subprocess.check_output("cmd args", shell=True)
retval=subprocess.check_output(["cmd","arg"])
retval=str(retval, 'utf-8') # Convert from byte string
```

✔ Lots of flexibility – recommended way to do it!

✘ Python >= 2.7

```
import commands
(status,retval) = commands.getstatusoutput("cmd args")
```

✔ Simple!

✘ Unix only

✘ Deprecated in Python 3

The easy way - Python2

```
import commands

exe = 'mysql --batch --silent -e "SELECT * FROM test" test'
(status, result) = commands.getstatusoutput(exe)

tuples = result.split("\n")

for tuple in tuples:
    fields = tuple.split("\t")
    for field in fields:
        print field, "   ",
    print
```

Why a standardized API?

- Databases generally provide **own APIs** to allow access from programming languages
  - e.g. C, Java, Perl, Python
- Proprietary APIs **all differ**
- Very **difficult to port** software between databases
- **Standardized APIs** have thus become available

PEP249: Python Database API Specification V2.0

www.python.org/dev/peps/pep-0249

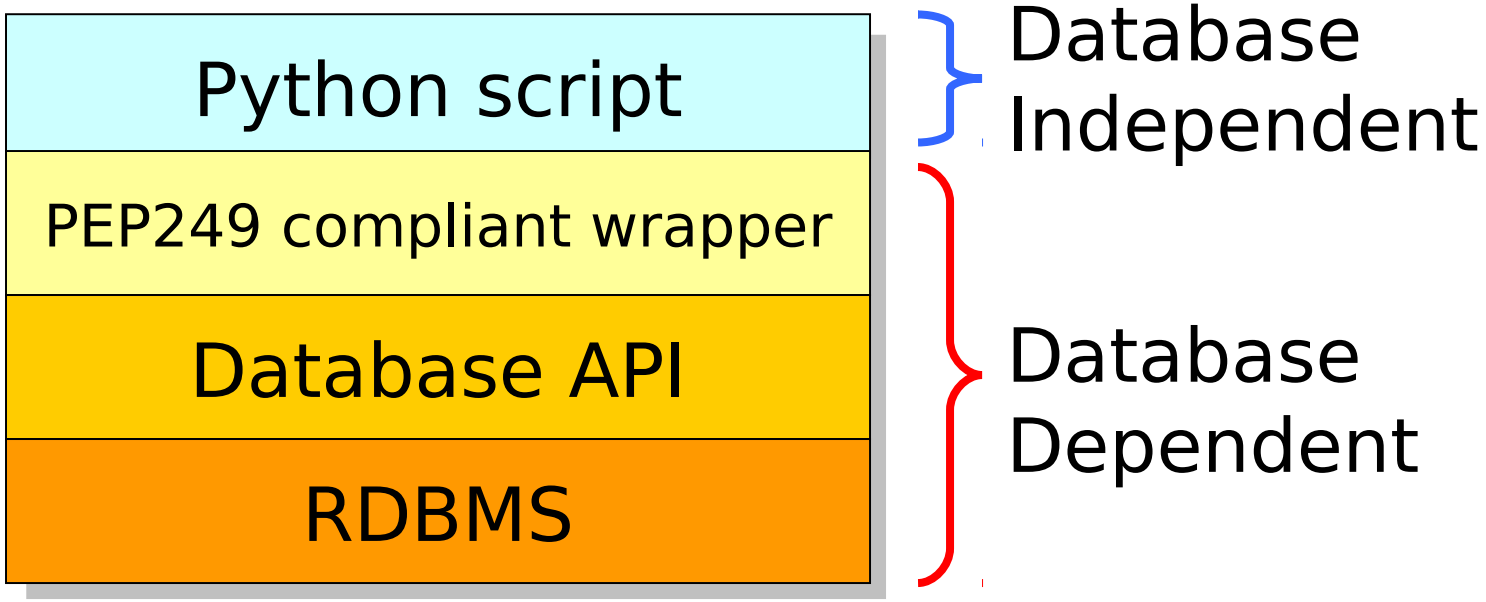
PEP249

- PEP249 is a **standard API** for Python
- Makes it easy to port Python scripts from one database to another

- PEP249 is a standard recommendation for API implementers – it is not software in itself.

API Architecture

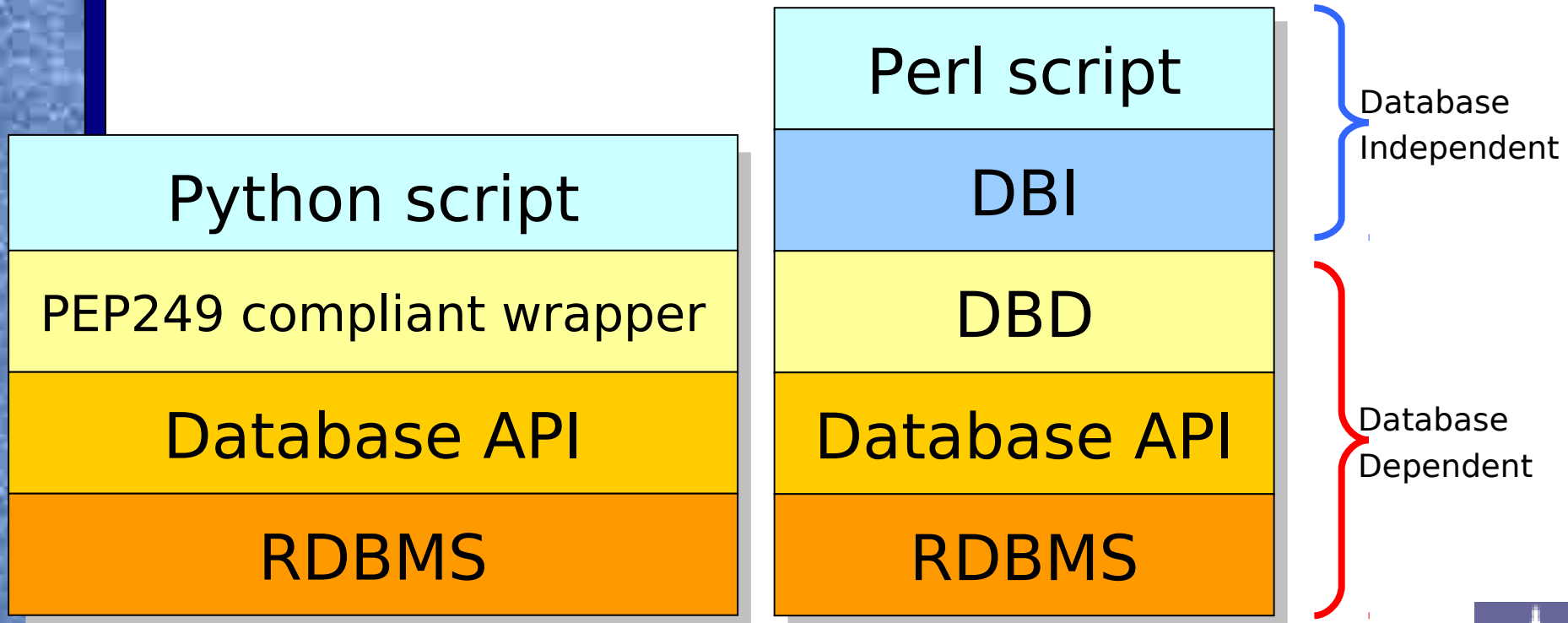
- Multi-layer design



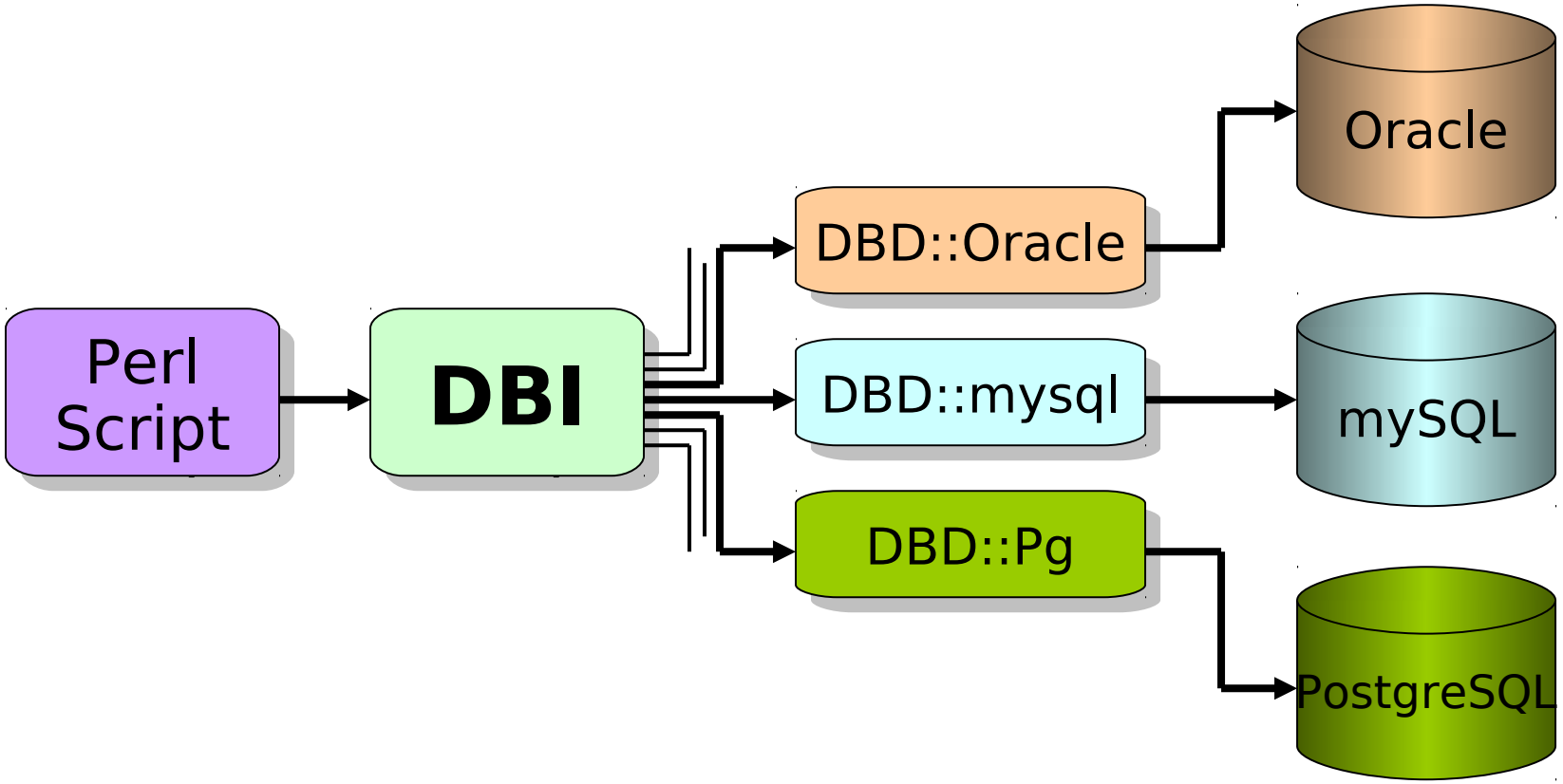
A comparison with DBI

PEP249 vs DBI Architecture

- Multi-layer design



DBI Architecture



ODBC – an alternative to PEP249

ODBC

- **ODBC** (Open DataBase Connectivity)
- Consortium of vendors in early 1990s
  - SQL Access Group
- October 1992 & 1993, **draft standard**:
  - 'Call Level Interface' ('CLI' - an API)
  - Never really adopted
- **Microsoft** *'embraced and extended'* it to create ODBC

Python and ODBC

- Various ODBC drivers for different databases including MySQL

- wiki.python.org/moin/ODBCDrivers

Using PEP249

SQL commands and their returns

- SQL commands other that SELECT have **zero rows** returned
  - creating a table
  - inserting a row e.g.
    ```
    INSERT INTO idac VALUES ('LYC_CHICK','P00698')
    ```
  - modifying a row
- SELECT statements can be guaranteed to have **one row** returned. e.g.
  ```
  SELECT count(*) from idac
  ```
- Others have **one or more rows** returned. e.g.
  ```
  SELECT * from idac
  ```

SQL commands and their returns

- Perl/DBI offers different methods depending on the returns:
  - `->do();` (none)
  - `->selectrow_array();` (single)
  - `->execute() / fetchrow_array();` (multiple)
- PEP249 has no such distinction
  - `.execute() / .fetchone()`

PEP249 Constructors

- The **connect** constructor

Parameter	Meaning
dsn	Data source name as string
user	Username (optional)
password	Password (optional)
host	Hostname (optional)
database	Database name (optional)

e.g.

```
db=connect(dsn='myhost:MYDB', user='guido', password='234$')
```

The MySQL driver is not fully compliant

```
import pymysql.cursors
db = pymysql.connect(host="localhost", port=3306,
                     user="amartin", db="test", passwd="2348")
```

PyMySQL

- Available in Python3
- Written in pure Python (so slower than some other options)
- Other options available...

PEP249 Methods

- **.close()**
  - Close the connection
- **.commit()**
  - Commit pending transactions
  - **Changes to a table will not appear until this is called!!!**
- **.rollback()**
  - Roll back pending transactions
- **.cursor()**
  - Obtain a database cursor

PEP249 Cursor Object

- Represent a database cursor
- Used to manage the context of a fetch operation.

PEP249 Cursor Attributes

- **.description** – returns a sequence of 7-item sequences
  - Each sequence represents one column

Item	Purpose
name	Name
type_code	Data type
display_size	
internal_size	
precision	
scale	
null_ok	

- **.rowcount**
  - Number of rows produced or modified by the last **execute()**; -1 on error (currently)
- **.arraysize**
  - Sets or shows the default number of rows to read

PEP249 Cursor Methods

- **.close()** – close the cursor
- **.execute(operation [, params])** – execute a SQL statement.
- **.fetchone()** – fetch the next row as a sequence
  - returns None when no more data
- **.fetchall()** – fetches all remaining rows
- **.fetchmany([size=cursor.arraysize])** – fetch the next set of rows as a sequence of sequences (default is set using cursor.arraysize)
  - returns an empty sequence when no more rows to fetch.

PEP249 Cursor

- The cursor is an iterator, so there are two ways of getting a line at a time:

```
data = cursor.fetchone()
while data is not None:
    pdb   = data[0]
    resol = data[1]
    name  = data[2]

    print (pdb, " ", resol, " ", name)
    data = cursor.fetchone()
```

```
for data in cursor:
    pdb   = data[0]
    resol = data[1]
    name  = data[2]

    print (pdb, " ", resol, " ", name)
```

A complete example

```
import pymysql.cursors

# Set parameters – ideally in some separate config module or file
dbname   = "mydb"
dbhost   = "localhost"
dbuser   = "myuser"

# Create SQL statement to find information for PDB entry 1DM5
sql = "select id, name, data from mytable where name = 'myname'”

# Connect to the database
db = pymysql.connect(host=dbhost, user=dbuser, db=dbname)

# Create a cursor and execute the SQL on it
cursor = db.cursor()
nrows  = cursor.execute(sql)

# Using the cursor as iterator
for row in cursor:
    id    = row[0]
    name  = row[1]
    data  = row[2]

    print (id, " ", name, " ", data)
```

Summary

- PEP249 provides a **standard API**
- It does not standardize the **SQL**
- Because **it is a recommendation**, not all drivers are fully compliant
- ODBC is an alternative
- Basic **4-step** process:
  - connect / cursor / execute / fetch
  - (fetch can also iterate over the cursor)

Other drivers

- Examples have been for Python 3
- Driver was PyMySQL
- Python2 drivers:
  - MySQLdb
  - mysql.connector
- Other Python3 drivers:
  - mysqlclient – low level, not PEP249
  - MySQLdb – sits on top of mysqlclient to provide PEP249 compatibility
  - Mysql.connector – PEP249, but not supported from Python 3.5

URLs

www.python.org/dev/peps/pep-0249/

dev.mysql.com/doc/connector-python/en/

stackoverflow.com/questions/23376103/python-3-4-0-with-mysql-database

(Examples use MySQLdb instead of mysql.connector – MySQLdb hasn't been ported to Python3)

ianhowson.com/a-quick-guide-to-using-mysql-in-python.html

Linux installation

Local user install

```
pip3 install --user PyMySQL
```

Install on Fedora/CentOS

```
dnf install python3-PyMySQL
```

Install on Ubuntu

```
apt-get install python3-PyMySQL
```