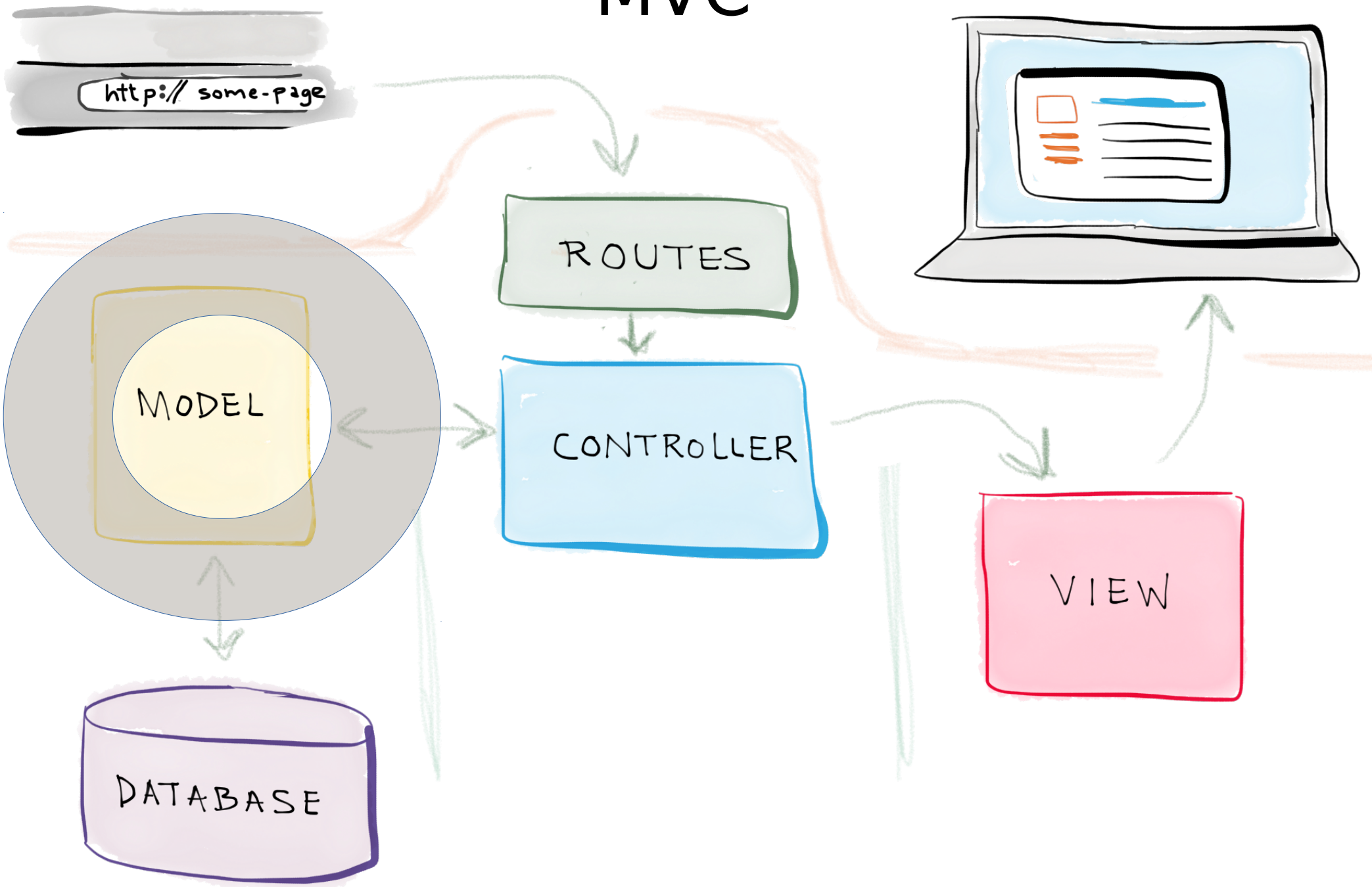
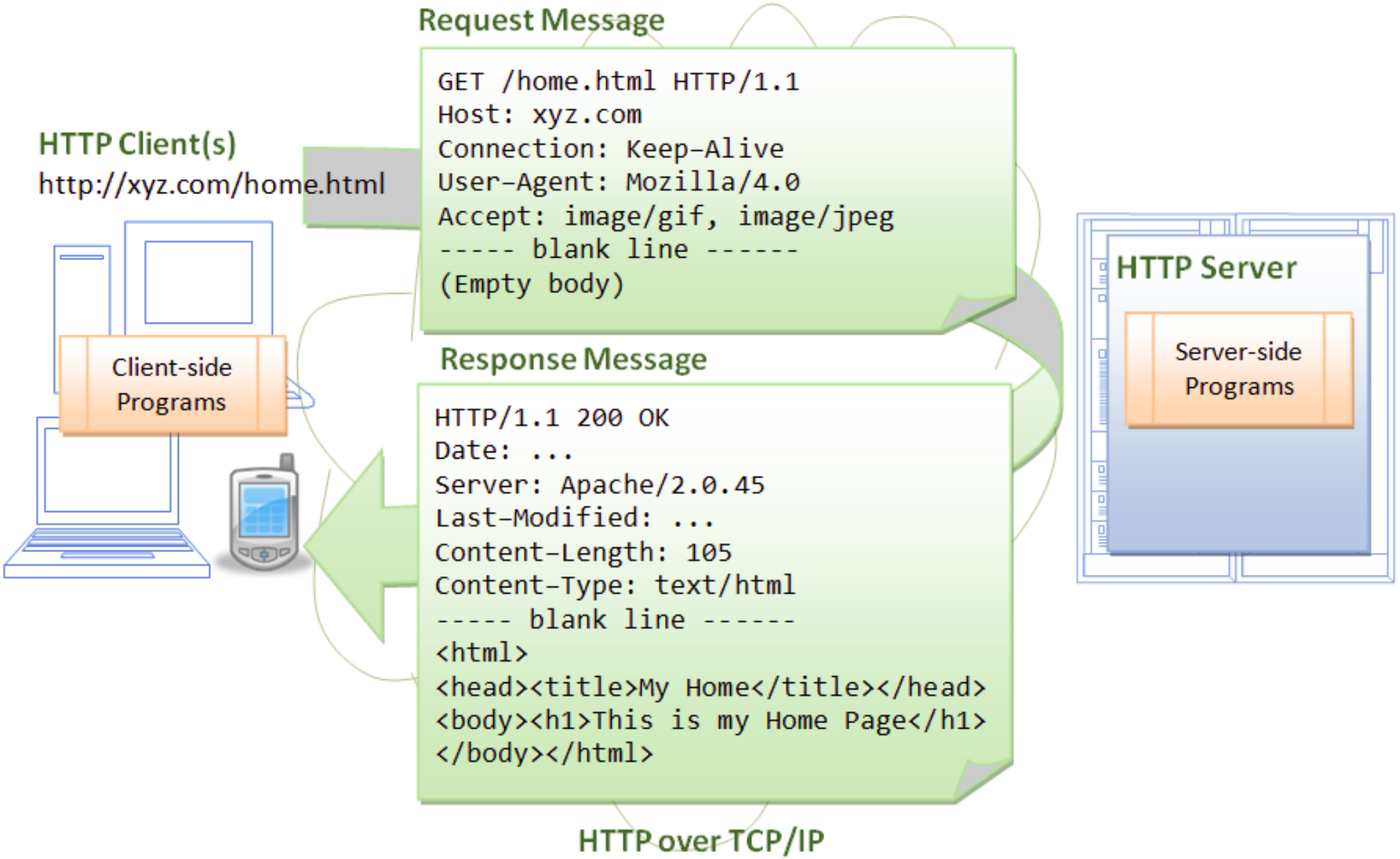
MVC



Review



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data

Databases(RDB)

- A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns.

- RDBs establish a well-defined relationship between database tables. Relations could be one to one, one to many, many to one, many to many implemented

- Tables communicate and share information using realtional operators for facilitating data searchability, organization and reporting.
  - "select" for data retrieval
  - "project" for identifing data attributes
  - "join" for combining relations.

- Advantages: extensibiity & security

Databases(RDBM)

- A relational database management system (RDBMS) is a database engine/system based on the relational model specified by Edgar F. Codd--the father of modern relational database design--in 1970.

- Most modern commercial and open-source database applications are relational in nature.

- The most important relational database features include an ability to use tables for data storage while maintaining and enforcing certain data relationships.

Databases(RDBM)

- Based on Mathematical Set Theory and first formulated and proposed in 1969 by E.F. Codd.

- In the relational model of a database, all data is represented in terms of:
  - tuples (row) or attribute (column)
  - grouped into relation (table).

- Declarative method

  - Directly stating what information the database contains and what information we want from it,

  - DBMS software takes care of efficient data storage, manipulation and retrieval.

Databases(Transaction)

- A transaction, in the context of a database, is a logical unit that is independently executed for data retrieval or updates.
- In relational databases, database transactions must be atomic, consistent, isolated and durable--summarized as the ACID acronym.

Databases(ACID -atomicity)

- All or nothing
- In all situations – crash, power failure, errors etc

Example:

Transferring funds from account A to B

A = A – x ---- 1. debiting A

B = B + x ---- 2. crediting B

Databases(ACID -consistency)

Any data written to the database must be valid according to all defined rules and constriants.

Example:

Integrity constraint A+B = 100

A = A – 10 => violates the constraint as A+B = 90

Databases(ACID - isolation)

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other.

Consider two transactions attempting to modify the same data. T1 transfers 10 from A to B. T2 transfers 10 from B to A. Combined, there are four actions:

```
T1: A = A - 10
T1: B = B + 10
```

```
T2: B = B - 10
T2: A = A  + 10
```

If these operations are performed in order, isolation is maintained, although T2 must wait. Consider what happens if T1 fails halfway through. The database eliminates T1's effects, and T2 sees only valid data.

By interleaving the transactions, the actual order of actions might be:

```
T1: A = A - 10
      T2: B = B - 10
            T2: A = A + 10
                  X T1: B = B + 10  T1 fails
```

Databases(ACID -durability)

The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

```
T1: A = A - 10
    T1: B = B + 10
        Power failure
        Persist changes to disk – T1 committed
```

```
T1: A = A - 10
     T1: B = B + 10
         Persist changes to disk – T1 committed
         Power failure
```

Databases(primary key)

A primary key is a special relational database table column (or combination of columns) designated to uniquely identify all table records.

A primary key's main features are:

- It must contain a unique value for each row of data.
- It cannot contain null values.

A primary key is either an existing table column or a column that is specifically generated by the database according to a defined sequence.

Almost all individuals deal with primary keys frequently but unknowingly in everyday life. e.g Ids, social security numbers

Databases(foreign key)

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.

It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

Databases(RI)

Referential integrity (RI) is a relational database concept, which states that table relationships must always be consistent.

In other words, any foreign key field must agree with the primary key that is referenced by the foreign key.

Thus, any primary key field changes must be applied to all foreign keys, or not at all. The same restriction also applies to foreign keys in that any updates (but not necessarily deletions) must be propagated to the primary parent key.

- Consider a stock database, which contains two tables:

  Customers and stocks
- Custid is set as **primary key** in customers and **foreign key** in stocks.
- **Referential integrity** is a standard that means any custid value in the customers table may not be edited without editing the corresponding value in the stocks table.

Databases(CRUD operations)

DDL (create/alter/drop):

structure of tables

DML (insert/delete/update):

manipulate data in tables

DQL (select):

retrieve data from table)

DTL (start transaction/commit/rollback/savepoint):

manage transactions

Databases(RI)

DEMO

SQLite: Introduction

- An embedded relational DBMS engine.
- The most widely-deployed DB engine. Used in browsers, embedded systems, Oss.
- Self-contained or standalone with very few dependencies
- serverless and zero-configuration as no separate server process to install, setup, configure, initialize, manage, and troubleshoot
- Transactional SQL engine (ACID)
- Supports concurrency - Allows multiple applications to access the same database at the same time.
- Has many bindings (APIs) to various programming Languages.
- Included in Python 2.5+ (as sqlite3 module)
- In-process C lib based SQL engine

SQLite: Introduction

SQLite has a full-featured SQL implementation

SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is open-source but it is not open-contribution.

SQLite is a compact library.

SQLite is a high-reliability storage solution.

Works well for

- embedded devices
- low to medium traffic web sites,
- client/server applications with server side databses
- internal, temporary, stand -in databases for educational, prototyping
- replacement for ad-hoc disk files
- data analysis

Python: SQL database API (DB-API)

Specification to encourage similarity between the Python modules that are used to access databases.

Connection objects

- represent a connection to a database
- are the interface to rollback and commit operations
- generate cursor objects

Methods:

- close()
- commit()
- rollback()
- cursor()

Python: SQL database API (DB-API)

Cursor objects

- represent a single SQL statement submitted as a string
- can be used to step through SQL statement results
- can execute

  DDL (create/alter: structure of tables),

  DML (insert/delete/update: data in tables)

  DQL (select: retrieve data from table)

Methods:

- execute(sql)
- close()
- fetchone()
- fetchmany(n)
- fetchall()

Python: SQL database API (DB-API)

Query results

- SQL select results returned to scripts as Python data
- Tables of rows are Python lists of tuples
- Field values within rows are normal Python objects
- An example query result: [('bob',38), ('emily',37)]

Python: SQLite3

```
import sqlite3

conn = sqlite3.connect('example.db')
c = conn.cursor()

# Create table
c.execute('''create table stocks (date text, trans text, symbol
text, qty real, price real)''')

# Insert a row of data
c.execute("""insert into stocks values ('2006-01-
05','BUY','AMZ',100,35.14)""")

for t in (('2006-03-28', 'BUY', 'IBM', 1000, 45.00),

          ('2006-04-05', 'BUY', 'MSOFT', 1000, 72.00),

          ('2006-04-06', 'SELL', 'IBM', 500, 53.00),

         ):

    c.execute('insert into stocks values (?,?,?,?,?)', t)
```

Python: SQLite3

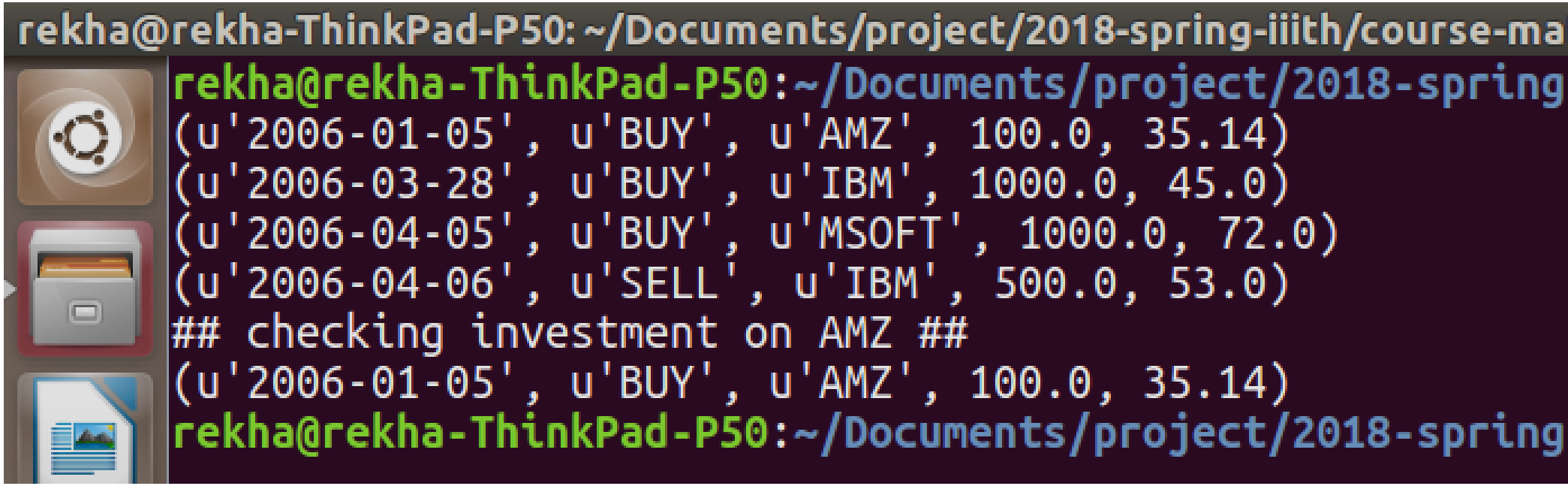
```
# run a query to get all records
c.execute('select * from stocks')
for row in c.fetchall():
    print row

# run a query
c.execute('select * from stocks where symbol=?', ('AMZ',))
row = c.fetchone()
print "## checking investment on AMZ ##"
print row

# commit/persist the changes to database
conn.commit()
conn.close()
```

Python: SQLite3

DEMO



Python: SQLite3- CL

Commands:

Create or open the database

$sqlite3 example.db

sqlite>.table

sqlite>select * from stocks;

Set the output sql file

Sqlite> .output stocks.sql

Dump all the sql code to consruct database in current state to output file

Sqlite> .dump

```
rekha@rekha-ThinkPad-P50:~/Documents/project/2018-spring-iiith/course-material/Rekha_slides/pyt
mo$ sqlite3 example.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .table
stocks
sqlite> select * from stocks;
2006-01-05|BUY|AMZ|100.0|35.14
2006-03-28|BUY|IBM|1000.0|45.0
2006-04-05|BUY|MSOFT|1000.0|72.0
2006-04-06|SELL|IBM|500.0|53.0
sqlite> .output stocks.sql
sqlite> .dump
```

Python: SQLite3- CL

Commands:

$ more stocks.sql

Edit output file to add/update records

$emacs stocks.sql &

Reconstruct database from output file

$sqlite3 test.db < stocks.sql

OR

$ sqlite3 test.db

Sqlite> .read stocks.sql

```
mo$ more stocks.sql
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE stocks (date text, trans text, symbol text, qty real, price real);
INSERT INTO "stocks" VALUES('2006-01-05','BUY','AMZ',100.0,35.14);
INSERT INTO "stocks" VALUES('2006-03-28','BUY','IBM',1000.0,45.0);
INSERT INTO "stocks" VALUES('2006-04-05','BUY','MSOFT',1000.0,72.0);
INSERT INTO "stocks" VALUES('2006-04-06','SELL','IBM',500.0,53.0);
COMMIT;
```