Lesson 12 – Regular Expressions

← Independent, Cryptic but Powerful (模糊的)

character based

↑ Cool tech thing, idea from 1960s

① Basics

>>> import re ← this is a must before using Regular Expression

Regular Expression Quick Guide

```
^         Matches the beginning of a line
$         Matches the end of the line
.         Matches any character
\s        Matches whitespace
\S        Matches any non-whitespace character
*         Repeats a character zero or more times
*?        Repeats a character zero or more times (non-greedy)
+         Repeats a character one or more times
+?        Repeats a character one or more times (non-greedy)
[aeiou]   Matches a single character in the listed set
[^XYZ]    Matches a single character not in the listed set
[a-z0-9]  The set of characters can include a range
(         Indicates where string extraction is to start
)         Indicates where string extraction is to end
```

The Regular Expression Module

- Before you can use regular expressions in your program, you must import the library using "import re"
- You can use re.search() to see if a string matches a regular expression, similar to using the find() method for strings
- You can use re.findall() to extract portions of a string that match your regular expression, similar to a combination of find() and slicing: var[5:10]

② re.search() ← Matching

Using re.search() like find()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print(line)
```

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print(line)
```

Using re.search() like startswith()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print(line)
```

③ "." ; "*" ; "\S" ; "^" ← Matching

Wild-Card Characters

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

^X.*:

Match the start of the line

Many times

Match any character

Fine-Tuning Your Match

Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks
```

^X-\S+:

Match the start of the line

One or more times

Match any non-whitespace character

---

④ Matching & Extracting Data → re.findall()

Matching and Extracting Data

- re.search() returns a True/False depending on whether the string matches the regular expression
- If we actually want the matching strings to be extracted, we use re.findall()

[0-9]+

One or more digits

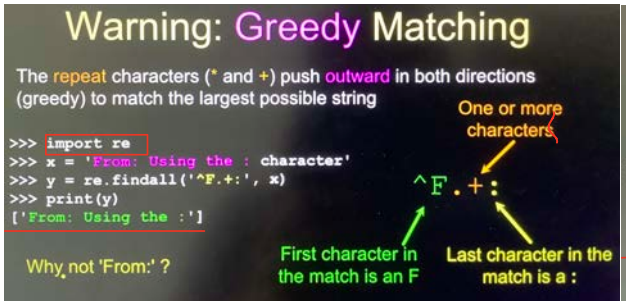
```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print(y)
['2', '19', '42']
```

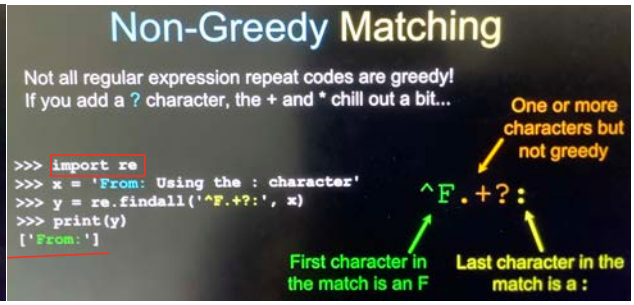
Matching and Extracting Data

When we use re.findall(), it returns a list of zero or more sub-strings that match the regular expression

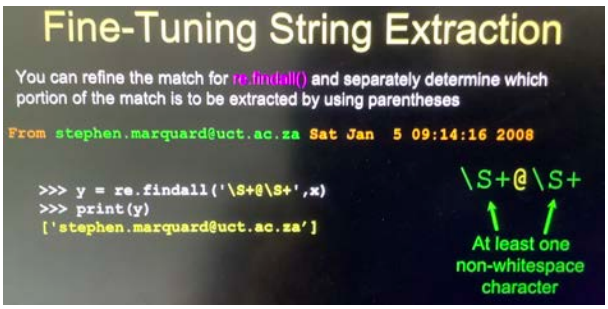
```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print(y)
['2', '19', '42']
>>> y = re.findall('[AEIOU]+',x)
>>> print(y)
[]
```

⑤ Warning: Greedy vs. Non-Greedy Matching

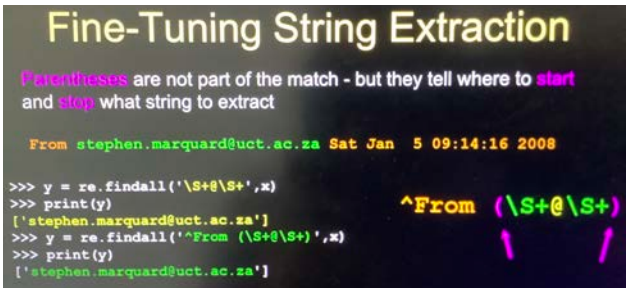




⑥ Fine-Tuning String Extraction (微調抓取)



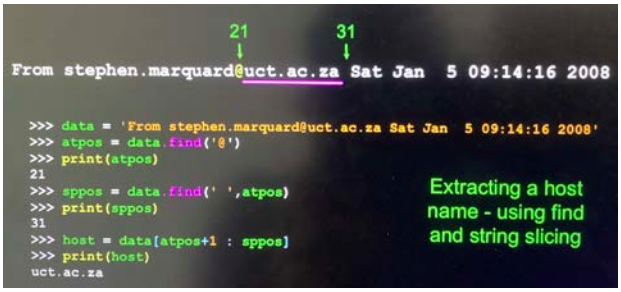
⑦ "( )" to indicate where to start and stop extraction (parentheses)



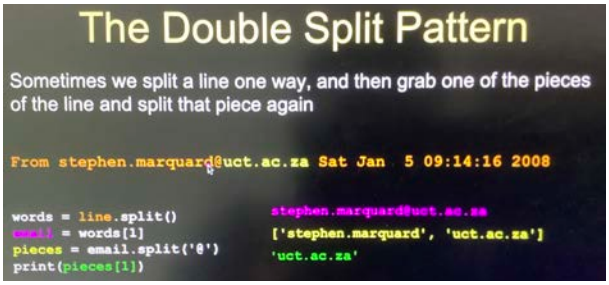
⑧ Example: String Parsing

※ Recommend

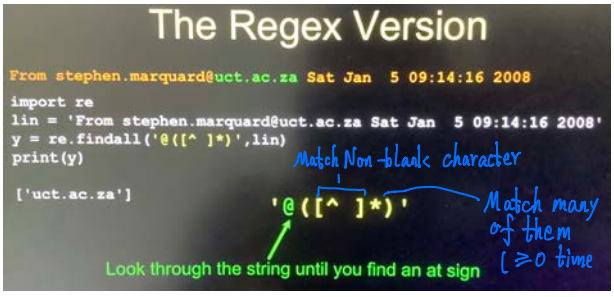
Method 1 → String & x.find()



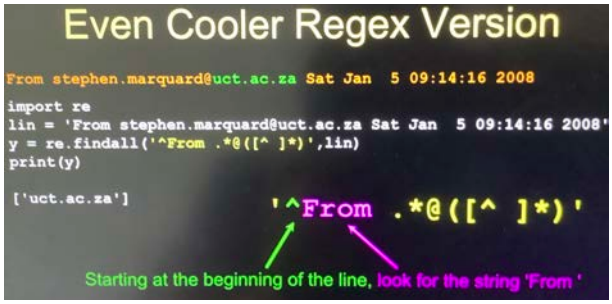
Method 2 → list & x.split( )



Method 3 → Regular Expression



Method 3' → Cooler & Better Regex version



↑ This is better than the left

Example : Spam Confidence

Spam Confidence

```
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 : continue
    num = float(stuff[0])
    numlist.append(num)
print('Maximum:', max(numlist))
```

```
python ds.py
Maximum: 0.9907
```

```
X-DSPAM-Confidence: 0.8475
```

(9) ~~X~~ Behave normally using "\" for a special character

Escape Character

If you want a special regular expression character to just behave normally (most of the time) you prefix it with '\'

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9.]+',x)
>>> print(y)
['$10.00']
```

At least one or more

\$[0-9.]+

A real dollar sign

A digit or period

(10) Exercise : Read through and parse a file with text and numbers. Then extract all the numbers in the file and compute the sum of the numbers.

Data Format

The file contains much of the text from the introduction of the textbook except that random numbers are inserted throughout the text. Here is a sample of the output you might see:

```
Why should you learn to write programs? 7746
12 1929 8827
Writing programs (or programming) is a very creative
7 and rewarding activity.  You can write programs for
many reasons, ranging from making your living to solving
8837 a difficult data analysis problem to having fun to helping 128
someone else solve a problem.  This book assumes that
everyone needs to know how to program ...
```

The sum for the sample text above is **27486**. The numbers can appear anywhere in the line. There can be any number of numbers in each line (including none).

Handling The Data

The basic outline of this problem is to read the file, look for integers using the **re.findall()**, looking for a regular expression of '**[0-9]+**' and then converting the extracted strings to integers and summing up the integers.

Method 1 - Classic Regex

Regex.py | Compact.py

```
import re
fname = input("Enter the file name:")
if len(fname) < 1: fname = "regex_sum_922639.txt"
handle = open(fname)
total = 0

for line in handle:
    line = line.rstrip()
    renum = re.findall('[0-9]+',line)
    # print(renum)
    for num in renum:
        num = int(num)
        total = total + num

print(total)
```

Method 2 - Closed form Regex

Regex.py | Compact.py

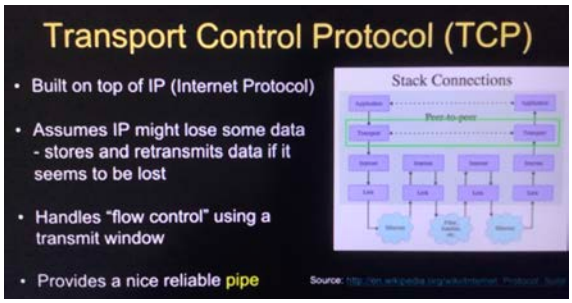
```
import re

fname = input("Enter the file name:")
if len(fname) < 1: fname = "regex_sum_922639.txt"
handle = open(fname)

print( sum( [ int(i) for i in re.findall('[0-9]+',handle.read()) ] ) )
```

Lesson 13 - Network Programming

talk to resources on the Internet

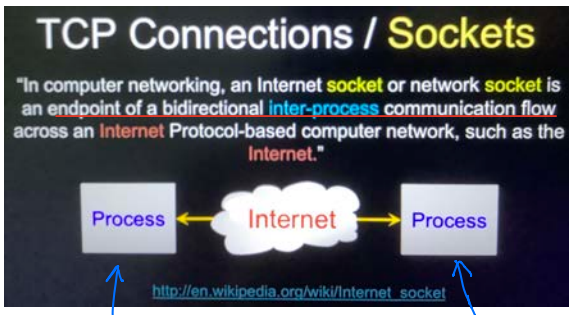
① Transport Control Protocol (TCP) (傳輸控制協議)

see more : www.net-intro.com



← Transport layer : Your computer assumes that it can make phone calls
← Internet layer : how to construct 15 or so hops to get packets back and forth
← link layer : how to get over one hop

② TCP Connections/Sockets (連接口, 插座)

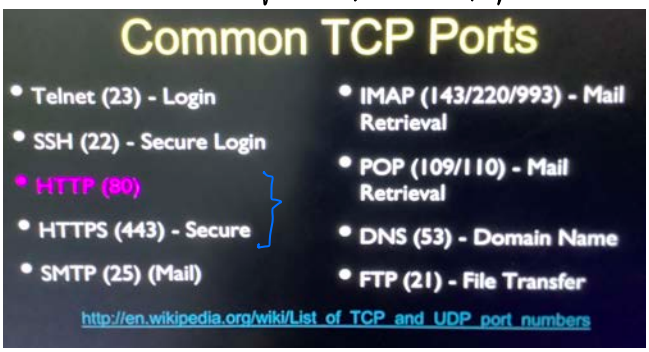


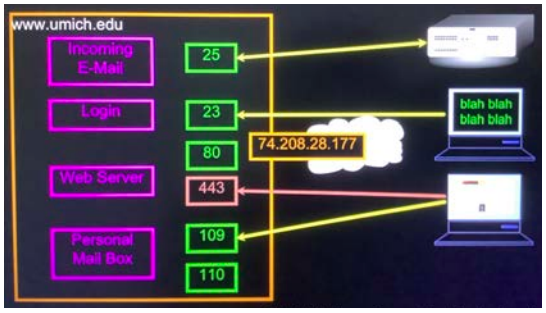
e.g.: Chrome

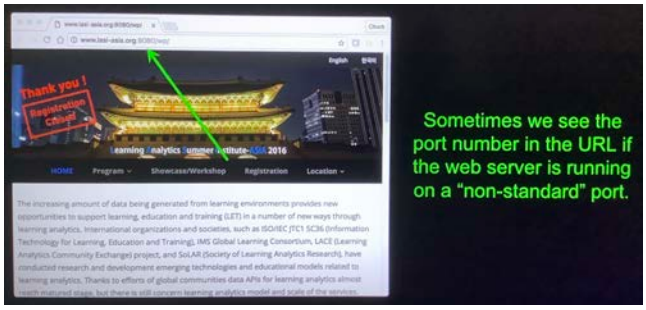
e.g.: web server

③ TCP Port Numbers (like extensions on phones) (端口)

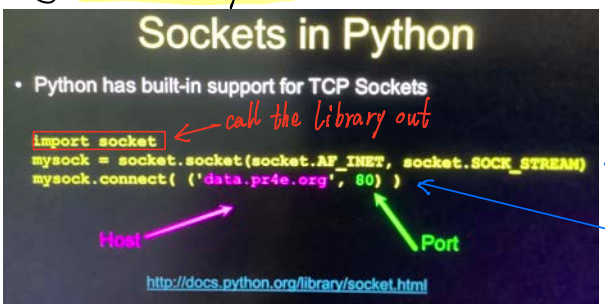
- A port is an application-specific or process-specific software communications endpoint
- It allows multiple networked applications to coexist on the same server
- There is a list of well-known TCP port numbers







④ Sockets in Python



call the library out

(not connected to Internet yet)

← "half open" : create/make a socket that goes across the Internet & it's a stream socket

make the phone call to this host and on that port

(phone No.) (phone extension)

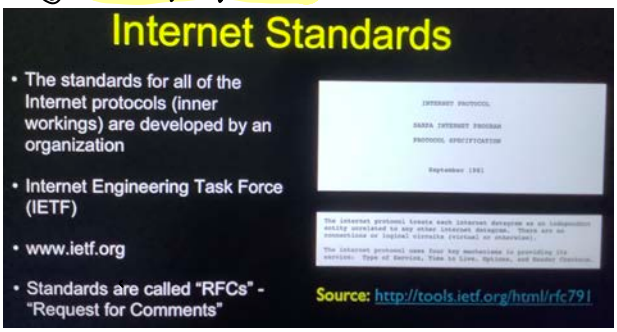
⑤ What's a Protocol? A set of rules that all parties follow so we can predict each other's behavior

⑥ ~~Application Protocols~~ (应用协议) (what to send & what to get back?)
· Mail · World Wide Web · file transfer ...

⑦ HTTP - Hypertext Transfer Protocol (超文本传输协议)
- The dominant Application Layer Protocol on the Internet
- Invented for the Web - to retrieve HTML, Images, Documents, etc.
- Extended to be data in addition to documents - RSS, Web Services, etc.. Basic Concept - Make a Connection - Request a document - Retrieve the Document - Close the Connection

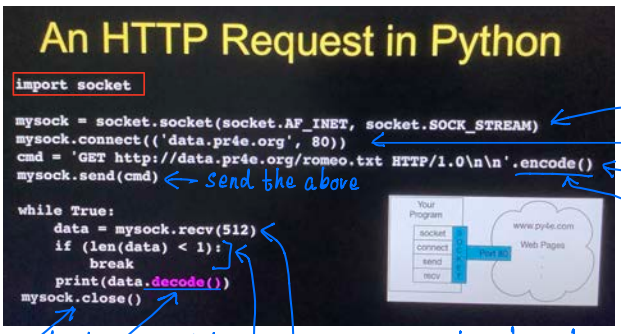
HTTP is the set of rules to allow browsers to retrieve web documents from servers on the Internet

⑧ Internet Standards



Let's write a Web Browser by Python

⑧ An HTTP Request in Python



data.decode() → transfer UTF-8 to Unicode that runs inside Python

make a stream-based socket

make phone call

we talk first; prepare what to send → "want to get XXX"

send the above

preparing the data to go across the Internet

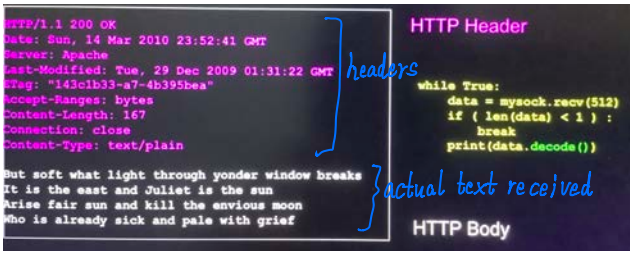
(There are strings in Unicode & have transfered to UTF-8)

decode outside data

receive 512 characters at a time

close the socket

if get zero character, it means the end of the stream & break



← the result

⑨ urllib library

↑ deal with like handling a file

e.g.:
```
>import urllib.request, urllib.parse, urllib.error
>fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
>for line in fhand:
    print(line.decode().strip())
```

This directly opens as 'urllib', may be blocked by website security

Get round by requesting as browser

```
>url = 'http://xxx'
>page = urllib.request.Request(url, headers=
                {'User-Agent': 'Mozilla/5.0'})
>fhand = urllib.request.urlopen(page)
```

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')

counts = dict()
for line in fhand:
    words = line.decode().split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1
print(counts)
```
urlwords.py

⑩ Web Scraping – "spidering the web"/"web crawling"

↓ Be careful about being blocked

- Pull data - particular social data - who link to who?
- Get your own data back out of some system that has no "export capability".
- Monitor a site for new information
- Spider the web to make a database for search engine

The Easy Way - BeautifulSoup (software)

e.g.: retrieve all tag 'a' links from source code

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl

ctx = ssl.create_default_context()
ctx.check_hostname=False
ctx.verify_mode=ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

Lesson 14 - Using Web Services

① "Wire protocol" - communicate between languages & agreement on a "Wire Format"

Two commonly used formats: XML and JSON

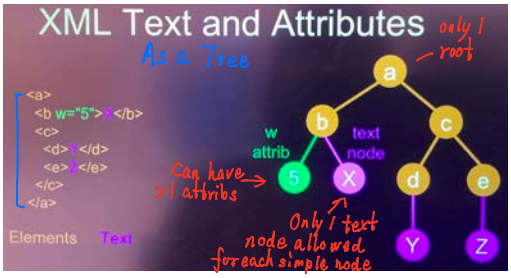
↓ word documents ↓ variety of

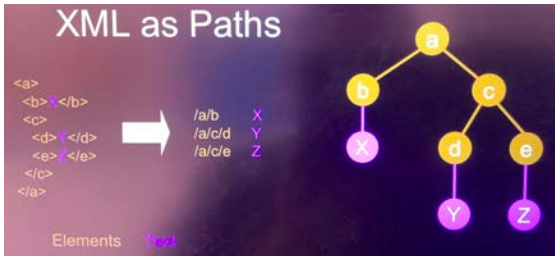
② XML - eXtensible Markup Language ⇒ share structured data

- Tags: Beginning & ending of elements
- Attributes: Keyword/value pairs on the opening tag of XML
- Serialize/De-Serialize - Convert data in one program into a common format can be stored and/or transmitted between systems in a programming language-independent manner

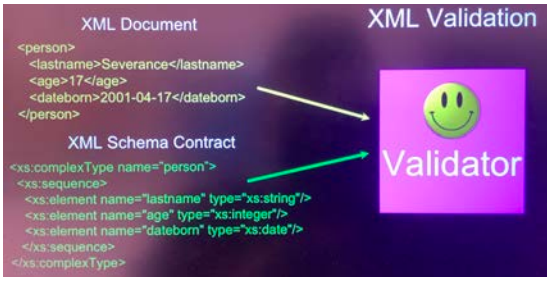
"< >"

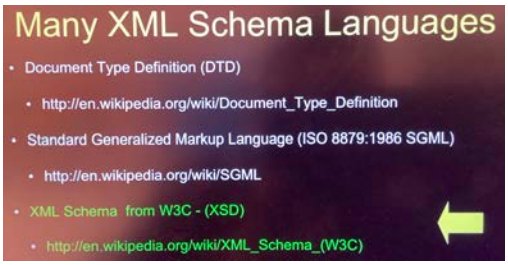
Line ends do NOT matter. White space is generally discarded on text elements. We indent only to be readable.



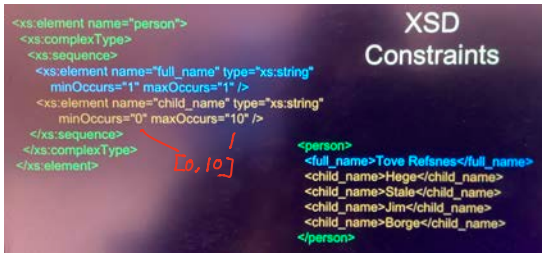


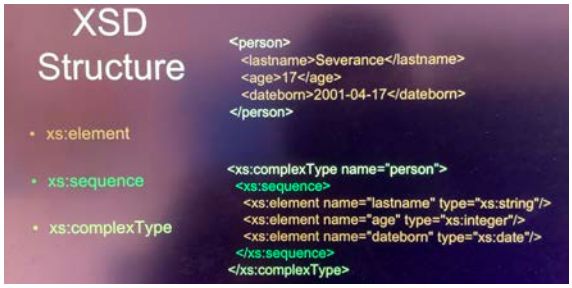
③ XML Schema → a language
- description of the legal format of an XML document
- often used to specify a "contract" between systems - "My system will only accept XML that conforms to this particular Schema"
- If a particular XML meets the specification of the Schema - it's said to be "validate"

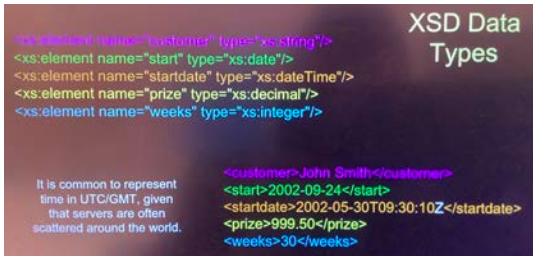




④ XSD XML Schema (W3C spec)
(the schema language we focus on)







⑤ XML in Python:
(talk to XML)

examples:

```
import xml.etree.ElementTree as ET
input = '''<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
        </user>
    </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))
for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

a list

```
import xml.etree.ElementTree as ET
data = '''<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print('Name:',tree.find('name').text)
print('Attr:',tree.find('email').get('hide'))
```

⑥ JSON - JavaScript Object Notation (curly brackets {} / square brackets [])

Better for pulling data & moving between systems — dictionary — list

JSON represents data as nested "lists" and "dictionaries" directly & easy to communicate

Dictionary of dictionaries

```
import json
data = '''{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}'''
info = json.loads(data)
print('Name:',info["name"])
print('Hide:',info["email"]["hide"])
```

Dictionary in this case

List of dictionaries

```
import json
input = '''[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''
info = json.loads(input)
print('User count:', len(info))
for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])
```

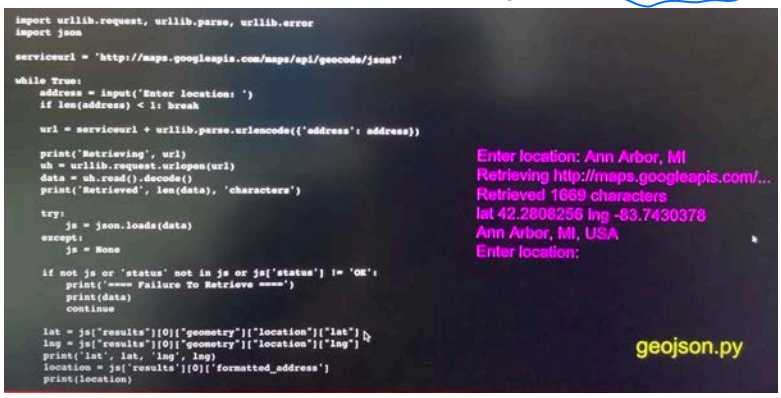
List in this case

Note: JSON is NOT a global standard, it's invented by a person and people love it. It is simplier than XML. (Douglas Crockford)

⑦ Services Oriented Approach (SOA) - Services

⑧ Web Services - GeoJSON "Application Program Interface" (API) - a contract for interaction

e.g.: Google maps API → Geocoding API http://developers.google.com/maps/documentation/geocoding/intro



⑨ API security and Rate Limiting → e.g.: Google Geocoding API has 2,500 requests/day limit

⑩ Web Services - Twitter API (in json) (optional) (a bit complicated)

Need a twitter account & set up a key

Lesson 15 – Object-Oriented Programming (terminology lecture)

① Object → a bit of self-contained Code and Data

② Class → a template for a kind of objects

③ dir(x) → show the capabilities of x, internal methods + .xx methods
↑ an object
use print(dir(x)) (ensure "setting" → "Soft wrap") to show all capabilities

④ (Seldomly used) Destructor in a class → def __del__(self):
XXX

⑤ Inheritance → the new class (child) has all the capabilities of the old class (parent) – and then some more

class childclass(parentclass):

⑥ OO programming is very structured approach to code reuse & group data and functionality together

---

Lesson 16 – Databases

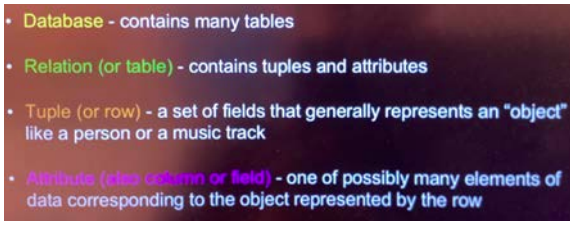
Part 1 – Relational Databases and SQLite

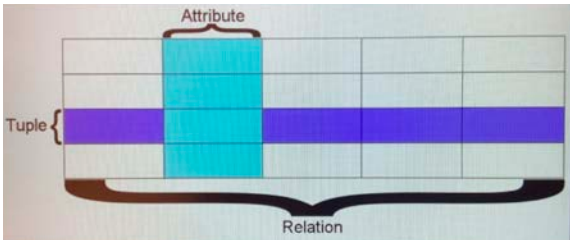
① Download & Install DB Browser – http://sqlitebrowser.org/

② History: Read data sequentially → Random Access (sorting not the best idea) & Simultaneously
the ideas lead to relational databases

③ Relational databases: Model data by storing rows and columns in tables.
The power → efficiently retrieve data and in particular where there are multiple tables and the relationships between those tables in the query.

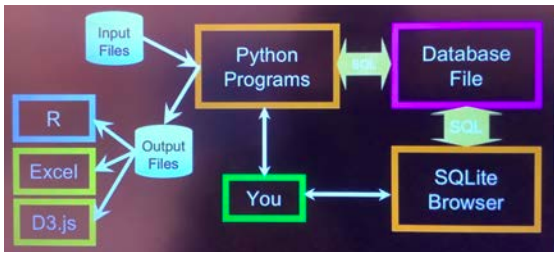
④ Terminology (like an Excel file with many sheets)



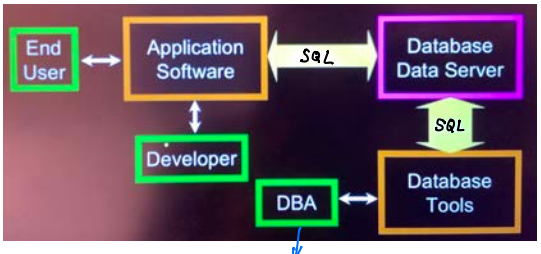


⑤ Structured Query Language (SQL) → the language we use to issue commands to the database
• create a table • retrieve some data • insert data • delete data (CRID)

⑥ Data analysis application flow:



⑦ Web Applications w/ Databases (e.g.: Amazon website)



Database administrator

⑧ Database Model (= database schema)
↳ the application of a data model when used in conjunction with a database management system.

⑨ Common Database Systems

- Three major Database Management Systems in wide use
  - Oracle - Large, commercial, enterprise-scale, very very tweakable
  - MySql - Simpler but very fast and scalable - commercial open source
  - SqlServer - Very nice - from Microsoft (also Access)
- Many other smaller projects, free and open source
  - HSQL, SQLite, Postgres, ...

↓ popular in lots of software & embedded in Python

Part 2 - Single Table SQL

① Get software "DB Browser for SQLite" ready

② Operation & Syntax for single table:

"New Database" → Table name → columns' names

"Browse Data" → "New record" (a symbol) → edit contents of cells

"Execute SQL" → <Codes line>: INSERT INTO {table name} (column name 1, column name 2, ...) VALUES (value 1, value 2, ...) ; ← the semicolon important,
(Insert) (can be lowercase)
Can repeat several times to add multiple info

→ execute in "Execute SQL"

"Execute SQL" → <Codes line>: DELETE FROM {table name} WHERE {column name = "value"} ;
(Delete) (can be lower case) (can repeat) → execute in "Execute SQL"

"Execute SQL" → <Codes line>: UPDATE {table name} SET {column name = new value} WHERE {column name = "value"};
(Update) (can be lower case) (can repeat) → execute in "Execute SQL"

"Execute SQL" → <Codes line>: SELECT * FROM {table name} WHERE {column name = "value"}
(Retrieve → Select) (all columns) (can be lower case) (can repeat) → execute in "Execute SQL"

SELECT * FROM {table name} ORDER BY {column name} DESC
optional, if reverse

③ Example of single table SQL:

```
Single Table SQL

CREATE TABLE "Users" ("name" TEXT, "email" TEXT)

INSERT INTO Users (name, email) VALUES ('Chuck', 'csev@umich.edu')
INSERT INTO Users (name, email) VALUES ('Colleen', 'cvl@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Sally', 'al@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Kristen', 'kf@umich.edu')

DELETE FROM Users WHERE email='ted@umich.edu'

UPDATE Users SET name="Charles" WHERE email='csev@umich.edu'

SELECT * FROM Users

SELECT * FROM Users WHERE email='csev@umich.edu'

SELECT * FROM Users ORDER BY email

SELECT * FROM Users ORDER BY name DESC
```

Part 3 - Complex Models (Multiple-tables)

① Database Design → art form of its own with particular skills and experience. It starts with pictures.

② Basic Rule: Do NOT put the same string data in twice - use a relationship instead

※ When there is one thing in the real world there should be one copy of that thing in the database. Columns with duplicate cells' contents are usually not treated as foundational notion.

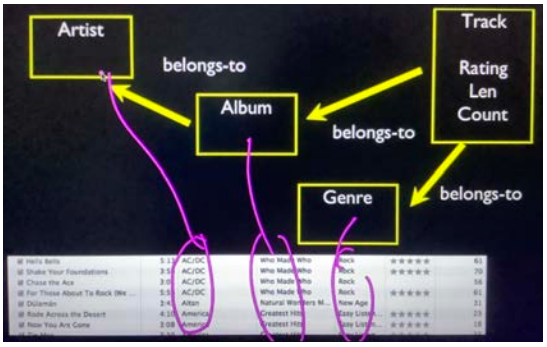
③ Project example:

Two questions: 1. Is the column of left an object or an attribute of another object?

2. Once we define objects, we need to define relationships between objects.

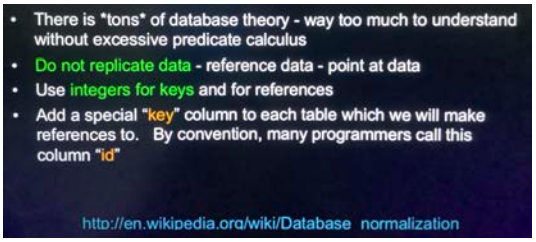
※ 1st step: Determine the foundational notion (usually string contents, "Track" in this case)

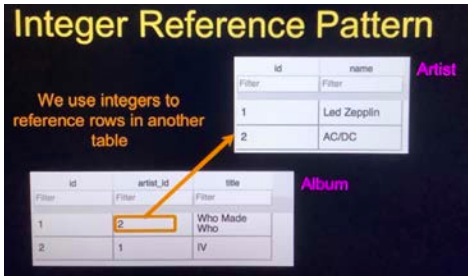
※ 2nd step: Grouping all columns. Columns with duplicate contents may have their own table. Columns with non-string contents are likely treated as attributes.



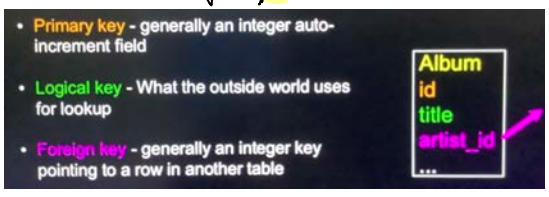
Part 4 - Represent relationships in a database

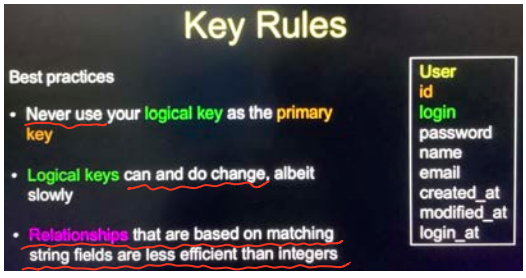
① Database Normalization (3NF)





② Three kinds of keys



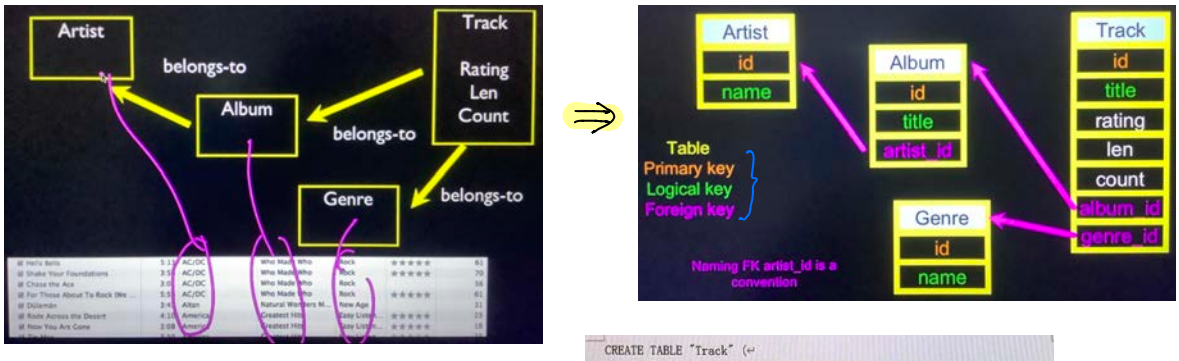


integer primary keys & foreign keys are very good.

Part 5 – Relationship Building in tables (with foreign keys)

① Construct a logical picture

Order of construction: Artist & Genre | Album | Track



② <Codes lines> Create table + Insert data + Join

```
Multi-Table SQL:

CREATE TABLE "Artist" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    "name" TEXT);

CREATE TABLE "Album" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    artist_id INTEGER,
    "title" TEXT);

CREATE TABLE "Genre" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    "name" TEXT);
```

auto primary keys: 1, 2, 3...

Order of construction: Artist & Genre | Album | Track

```
CREATE TABLE "Track" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL UNIQUE,
    album_id INTEGER, genre_id INTEGER, len INTEGER, rating INTEGER,
    "title" TEXT, "count" INTEGER);

INSERT INTO Artist (name) VALUES ('Led Zeppelin');
INSERT INTO Artist (name) VALUES ('AC/DC');

INSERT INTO Genre (name) VALUES ('Rock');
INSERT INTO Genre (name) VALUES ('Metal');

INSERT INTO Album (title, artist_id) VALUES ('Who Made Who', 2);
INSERT INTO Album (title, artist_id) VALUES ('IV', 1);

INSERT INTO Track (title, rating, len, count, album_id, genre_id)
    VALUES ('Black Dog', 5, 297, 0, 2, 1);
INSERT INTO Track (title, rating, len, count, album_id, genre_id)
    VALUES ('Stairway', 5, 482, 0, 2, 1);
INSERT INTO Track (title, rating, len, count, album_id, genre_id)
    VALUES ('About to Rock', 5, 313, 0, 1, 2);
INSERT INTO Track (title, rating, len, count, album_id, genre_id)
    VALUES ('Who Made Who', 5, 207, 0, 1, 2);

SELECT Album.title, Artist.name FROM Album JOIN Artist
    ON Album.artist_id = Artist.id;

SELECT Album.title, Album.artist_id, Artist.id, Artist.name
    FROM Album JOIN Artist ON Album.artist_id = Artist.id;

SELECT Track.title, Track.genre_id, Genre.id, Genre.name
    FROM Track JOIN Genre;

SELECT Track.title, Genre.name FROM Track JOIN Genre
    ON Track.genre_id = Genre.id;

SELECT Track.title, Artist.name, Album.title, Genre.name
FROM Track JOIN Genre JOIN Album JOIN Artist
    ON Track.genre_id = Genre.id AND Track.album_id = Album.id
    AND Album.artist_id = Artist.id;
```

No need to repeat

linked to 'AC/DC' (original data)

"JOIN" (see next part)

Album → Artist hide key

Album → Artist with keys

Track → Genre hide keys

Track → Genre with keys

Track → all other tables

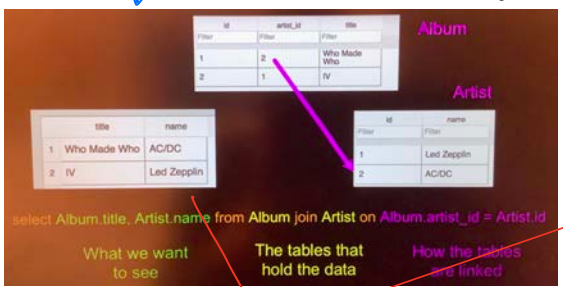
These are link tables we want to output

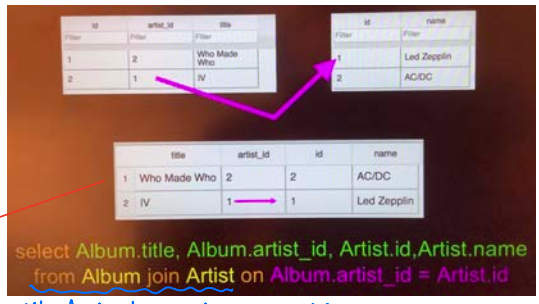
Part 6 – Join across tables "JOIN" to output tables

① JOIN operation: links across several tables as part of a select operation

Must tell the JOIN how to use the keys that make the connection between the tables using an ON clause — this manually control the combinations of linkage.

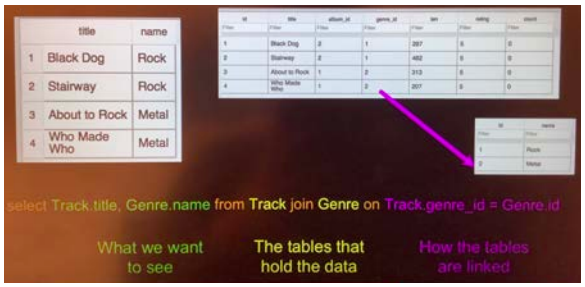
Syntax

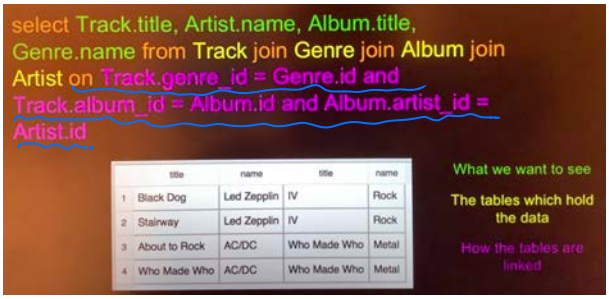




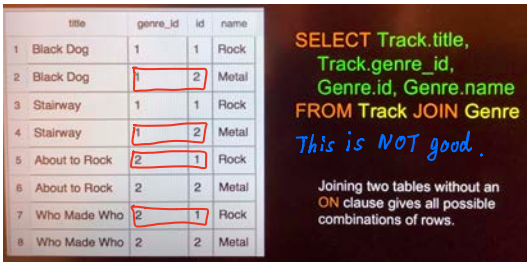
(the foreign keys are in Album table)

Note there is difference in output table, sometimes we want a linked table showing keys.





② ON Clause is Important: If no "ON" clause → will give all possible combinations of rows & cause mismatch

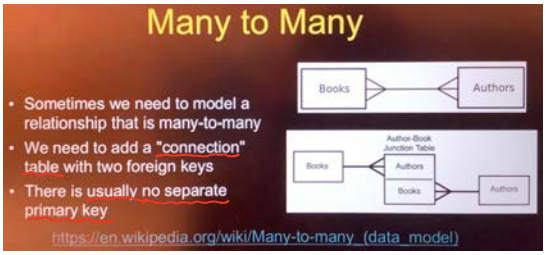


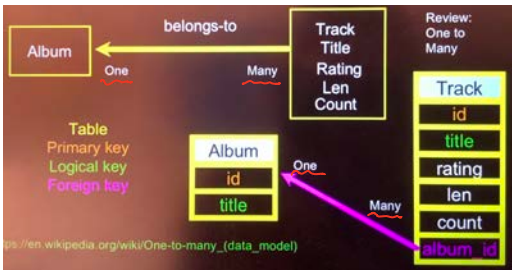
Do include "ON" clause to avoid mismatch

Part 7 - Many to Many Relationships

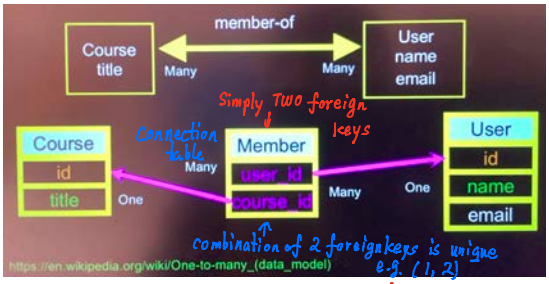
① Many to One Relationships are discussed previously

② Many to Many Relationships





③ Project as an example



Can regard this as the "primary" key

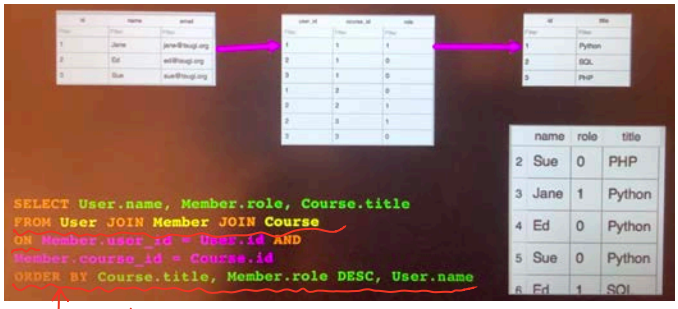
Like a paired Tuple

④ <Code lines>

```
Many-Many Relationship

CREATE TABLE User (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE,
    email   TEXT
) ;

CREATE TABLE Course (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title   TEXT UNIQUE
) ;

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role        INTEGER,
    PRIMARY KEY (user_id, course_id)
) ;

INSERT INTO User (name, email) VALUES ('Jane', 'jane@tsugi.org');
INSERT INTO User (name, email) VALUES ('Ed', 'ed@tsugi.org');
INSERT INTO User (name, email) VALUES ('Sue', 'sue@tsugi.org');

INSERT INTO Course (title) VALUES ('Python');
INSERT INTO Course (title) VALUES ('SQL');
INSERT INTO Course (title) VALUES ('PHP');

INSERT INTO Member (user_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 1, 0);

INSERT INTO Member (user_id, course_id, role) VALUES (1, 2, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 2, 1);

INSERT INTO Member (user_id, course_id, role) VALUES (2, 3, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 3, 0);

SELECT User.name, Member.role, Course.title
  FROM User JOIN Member JOIN Course
  ON Member.user_id = User.id AND Member.course_id = Course.id
  ORDER BY Course.title, Member.role DESC, User.name
```

person type

person type



order clause

Summary & Not covered SQL Topics

Complexity Enables Speed

- Complexity makes speed possible and allows you to get very fast results as the data size grows
- By normalizing the data and linking it with integer keys, the overall amount of data which the relational database must scan is far lower than if the data were simply flattened out
- It might seem like a tradeoff - spend some time designing your database so it continues to be fast when your application is a success

Additional SQL Topics

- Indexes improve access performance for things like string fields
- Constraints on data - (cannot be NULL, etc..)
- Transactions - allow SQL operations to be grouped and done as a unit

Summary

- Relational databases allow us to scale to very large amounts of data
- The key is to have one copy of any data element and use relations and joins to link the data to multiple places
- This greatly reduces the amount of data which much be scanned when doing complex operations across large amounts of data
- Database and SQL design is a bit of an art form

Final Part - SQLite examples with Python

① Many to One example - Tracks (tracks.py) from ipod's "Library.xml"

```
import xml.etree.ElementTree as ET
import sqlite3

conn = sqlite3.connect('trackdb.sqlite')
cur = conn.cursor()

# Make some fresh tables using executescript()
cur.executescript('''
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Album;
DROP TABLE IF EXISTS Track;

CREATE TABLE Artist (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE
);

CREATE TABLE Album (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    artist_id  INTEGER,
    title   TEXT UNIQUE
);

CREATE TABLE Track (
    id  INTEGER NOT NULL PRIMARY KEY
        AUTOINCREMENT UNIQUE,
    title TEXT  UNIQUE,
    album_id  INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
);
''')
```

The file NOT necessarily exists

Drop old

constrain - blowup signal

```
fname = input('Enter file name: ')
if ( len(fname) < 1 ) : fname = 'Library.xml'

# <key>Track ID</key><integer>369</integer>
# <key>Name</key><string>Another One Bites The Dust</string>
# <key>Artist</key><string>Queen</string>
def lookup(d, key):
    found = False
    for child in d:
        if found : return child.text
        if child.tag == 'key' and child.text == key :
            found = True
    return None

stuff = ET.parse(fname)
all = stuff.findall('dict/dict/dict')
print('Dict count:', len(all))
```

handle xml

```
for entry in all:
    if ( lookup(entry, 'Track ID') is None ) : continue

    name = lookup(entry, 'Name')
    artist = lookup(entry, 'Artist')
    album = lookup(entry, 'Album')
    count = lookup(entry, 'Play Count')
    rating = lookup(entry, 'Rating')
    length = lookup(entry, 'Total Time')

    if name is None or artist is None or album is None :
        continue

    print(name, artist, album, count, rating, length)

    cur.execute('''INSERT OR IGNORE INTO Artist (name)
        VALUES ( ? )''', ( artist, ) )
    cur.execute('SELECT id FROM Artist WHERE name = ? ', (artist, ))
    artist_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Album (title, artist_id)
        VALUES ( ?, ? )''', ( album, artist_id ) )
    cur.execute('SELECT id FROM Album WHERE title = ? ', (album, ))
    album_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Track
        (title, album_id, len, rating, count)
        VALUES ( ?, ?, ?, ?, ? )''',
        ( name, album_id, length, rating, count ) )

conn.commit()
```

avoid blowup

the frequency can be changed

List of lists in this case ↓

② Many to Many example - "Course - User" (roster.py) with Json file

```
import json
import sqlite3

conn = sqlite3.connect('rosterdb.sqlite')
cur = conn.cursor()

# Do some setup
cur.executescript('''
DROP TABLE IF EXISTS User;
DROP TABLE IF EXISTS Member;
DROP TABLE IF EXISTS Course;

CREATE TABLE User (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name   TEXT UNIQUE
);

CREATE TABLE Course (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title  TEXT UNIQUE
);

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role        INTEGER,
    PRIMARY KEY (user_id, course_id)
)
''')

fname = input('Enter file name: ')
if len(fname) < 1:
    fname = 'roster_data_sample.json'

# [
#   [ "Charley", "si110", 1 ],
#   [ "Mea", "si110", 0 ],

str_data = open(fname).read()
json_data = json.loads(str_data)
```

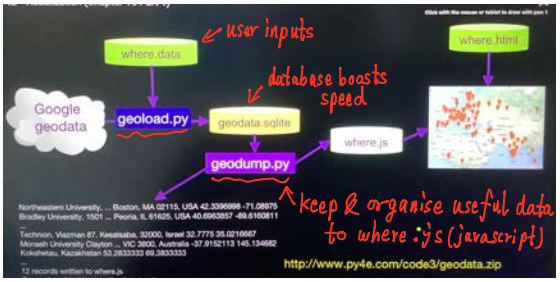
data sample

```
for entry in json_data:

    name = entry[0];
    title = entry[1];

    print((name, title))

    cur.execute('''INSERT OR IGNORE INTO User (name)
        VALUES ( ? )''', ( name, ) )
    cur.execute('SELECT id FROM User WHERE name = ? ', (name, ))
    user_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Course (title)
        VALUES ( ? )''', ( title, ) )
    cur.execute('SELECT id FROM Course WHERE title = ? ', (title, ))
    course_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Member
        (user_id, course_id) VALUES ( ?, ? )''',
        ( user_id, course_id ) )

conn.commit()
```

the frequency can be changed

Lecture 17 - Data Visualization → Personal Data Mining

Part 1 - Geodata
- → Make a Google Map from user entered data
- → Use the Google Geodata API
- → Caches data in a database to avoid rate limiting and allow restarting
- → Visualized in a browser using the Google Maps API.



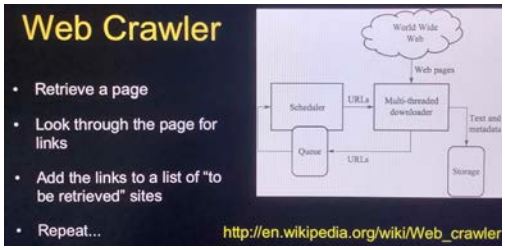
The real codes of the two .py files are too advanced, skip here.

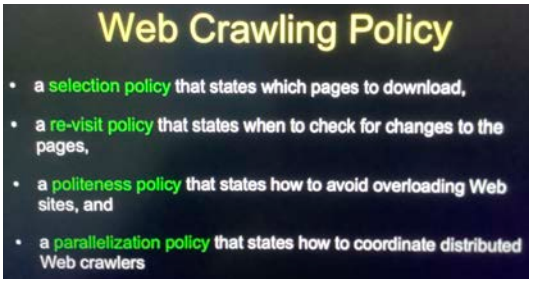
Database is useful for reconnecting & working from the cut point.

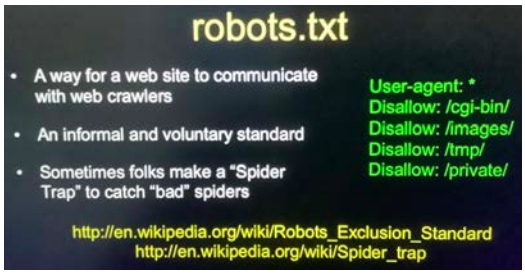
Part 2 - Page Rank (Google Rank)
- → write a simple web page crawler
- → compute a simple version of Google Page Rank Algorithm
- → visualize the resulting network

- Web crawling
- Index building
- Searching

①







② Index Building