

Lesson 12 - Regular Expressions

← Independent, Cryptic but Powerful

character based

↑ Cool tech thing, idea from 1960s

(模式识别)

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()`

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

Using `re.search()` like `startswith()`

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

③ ". " ; "*" ; "\S" ; "\A" ← 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-DSI-Result: Innocent
X-DSI-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain

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-DSI-Result: Innocent
X-Plane is behind schedule: two weeks

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)
[]
```

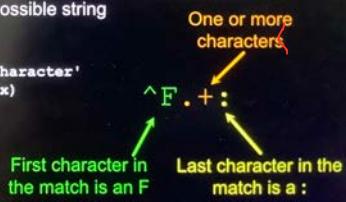
⑤ Warning: Greedy vs. Non-Greedy Matching

Warning: Greedy Matching

The repeat characters (*) and (+) push outward in both directions (greedy) to match the largest possible string

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print(y)
```

Why not 'From'?

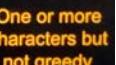


Non-Greedy Matching

Not all regular expression repeat codes are greedy! If you add a ? character, the + and * chill out a bit...

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print(y)
```

First character in the match is an F Last character in the match is a :

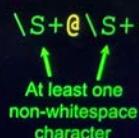


⑥ Fine-Tuning String Extraction

Fine-Tuning String Extraction

You can refine the match for `re.findall()` and separately determine which portion of the match is to be extracted by using parentheses

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
>>> y = re.findall('(\S+@\S+)',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```



Fine-Tuning String Extraction

Parentheses are not part of the match - but they tell where to start and stop what string to extract

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
>>> y = re.findall('(\S+@\S+)',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From (\S+@\S+)',x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

⑦ Example: String Parsing

Method 1 → String & `x.find()`

```
21           31
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> spos = data.find(' ',atpos)
>>> print(spos)
31
>>> host = data[atpos+1 : spos]
>>> print(host)
uct.ac.za
```

Extracting a host name - using find and string slicing

Method 2 → List & `x.split()`

The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

Method 3 → Regular Expression

The Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('@([^\s]+)',lin)
print(y)
['uct.ac.za']

Match Non-blank character
'@([^\s]+)'
Match many of them (>= 0 time)
Look through the string until you find an at sign
```

Even Cooler Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*(@[^\s]+)',lin)
print(y)
['uct.ac.za']

Starting at the beginning of the line, look for the string 'From '
'^(From .*(@[^\s]+))'
```

↑ This is better than the left

Example: Spam Confidence

```

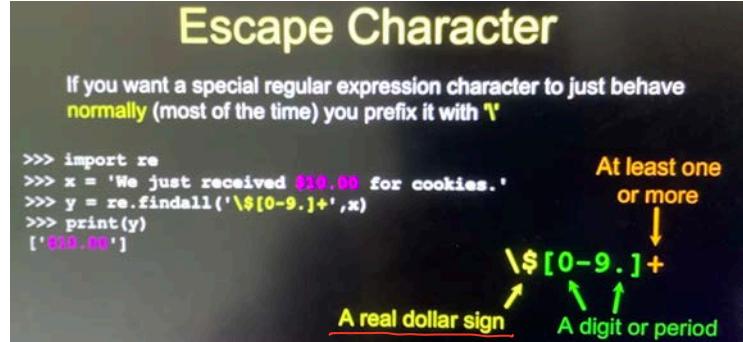
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1: continue
    num = float(stuff[0])
    numlist.append(num)
print('Maximum:', max(numlist))

X-DSPAM-Confidence: 0.8475

```

Spam Confidence
python ds.py
Maximum: 0.9907

- ⑦ Behave normally using “\”
for a special character



- 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

```

R0fDesktop|pyfile — Atom
Find Packages Help
Regex.py Compact.py
1 import re
2 fname = input("Enter the file name:")
3 if len(fname) < 1: fname = "regex_sum_922639.txt"
4 handle = open(fname)
5 total = 0
6
7 for line in handle:
8     line = line.rstrip()
9     renum = re.findall('[0-9]+',line)
10    for num in renum:
11        num = int(num)
12        total = total + num
13
14 print(total)
15
16

```

Method 2 - Closed form Regex

```

INRUR/Desktop|pyfile — Atom
Find Packages Help
Regex.py Compact.py
1 import re
2
3 fname = input("Enter the file name:")
4 if len(fname) < 1: fname = "regex_sum_922639.txt"
5 handle = open(fname)
6
7 print( sum( [ int(i) for i in re.findall('[0-9]+',handle.read()) ] ) )
8

```

Lesson 13 - Network Programming

talks to resources on the Internet

① Transport Control Protocol (TCP) (传输控制协议)

see more: www.net-intro.com

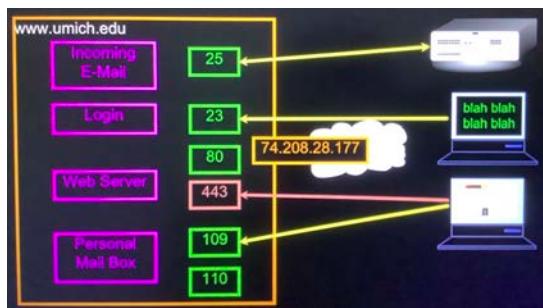
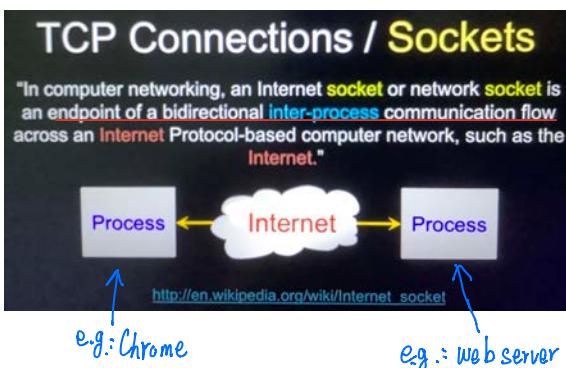
Transport Control Protocol (TCP)

- Built on top of IP (Internet Protocol)
- Assumes IP might lose some data
 - stores and retransmits data if it seems to be lost
- Handles "flow control" using a transmit window
- Provides a nice reliable pipe

Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

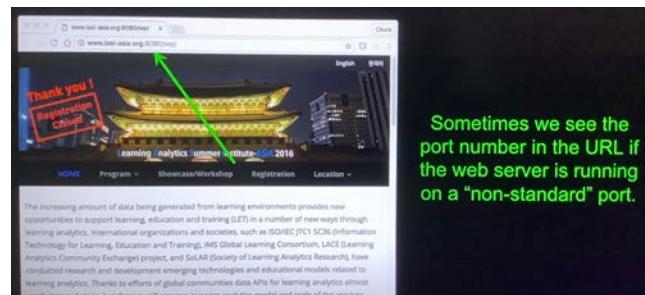
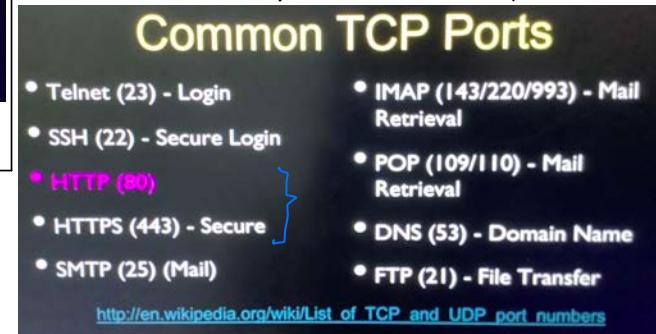
↳ Transport layer: Your computer assumes that it can make phone calls
↳ Internet layer: how to construct 15 or so hops to get packets
↳ link layer: how to get over one hop back and forth

② TCP Connections / Sockets (套接字, 插座)



③ TCP Port Numbers (类似于电话机上的插孔)

- 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

Sockets in Python

- Python has built-in support for TCP Sockets

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect( ('data.pr4.org', 80) )
```

Host → Port

http://docs.python.org/library/socket.html

(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

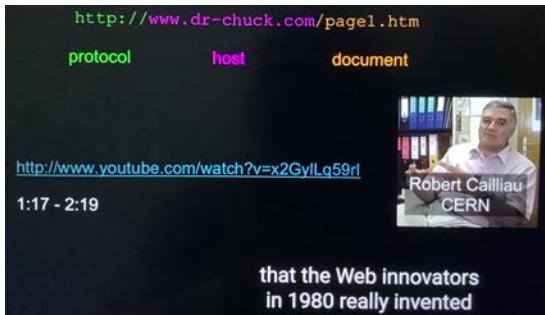
⑥ Application Protocols (应用协议) (What to send & what to get back?)

- 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

Internet Standards

- The standards for all of the Internet protocols (inner workings) are developed by an organization
- Internet Engineering Task Force (IETF)
- www.ietf.org
- Standards are called "RFCs" - "Request for Comments"

INTERNET PROTOCOL
DATA INTERNET PROGRAM
PROTOCOL SPECIFICATION
September 1981

The Internet protocol leaves much Internet design as an implementation detail unrelated to any other Internet layer. There are no connections or logical circuits (virtual or otherwise).
The Internet protocol uses four key mechanisms to provide its services. One of these mechanisms is the Request for Comments (RFC).

Source: <http://tools.ietf.org/html/rfc791>

Let's write a Web Browser by Python

⑨ An HTTP Request in Python

An HTTP Request in Python

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect('data.pr4e.org', 80)
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd) ← send the above

while True:
    data = mysock.recv(512) ← receive 512 characters at a time
    if len(data) < 1:
        break
    print(data.decode())
    mysock.close() ← close the socket
```

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"

preparing the data to go across the Internet
(There are strings in Unicode & have transferred to UTF-8)

```
HTTP/1.1 200 OK
Date: Sun, 14 Mar 2010 23:52:41 GMT
Server: Apache
Last-Modified: Tue, 29 Dec 2009 01:31:22 GMT
Etag: "143c1b33-a7-4b395bea"
Accept-Ranges: bytes
Content-Length: 167
Connection: close
Content-Type: text/plain

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

← the result

} actual text received
HTTP Body

- 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)
- urwords.py**
- ⑩ Web Scraping - "spidering the web" / "web crawling"
 ↓
Be careful about being blocked
- Pull data - particular social data - who links 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

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 ↓

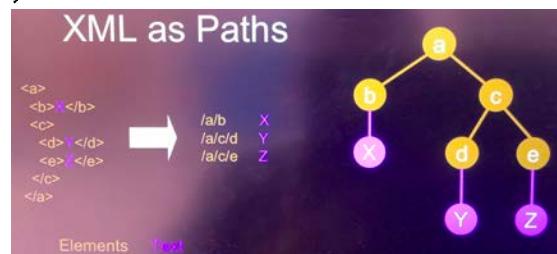
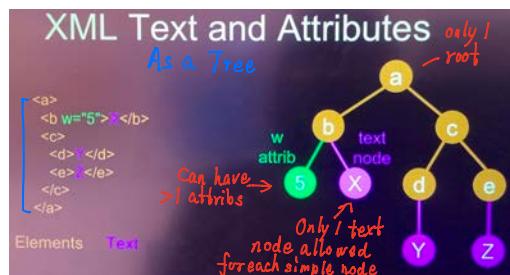
- ② XML - eXtensible Markup Language ⇒ share structured data



"< >"

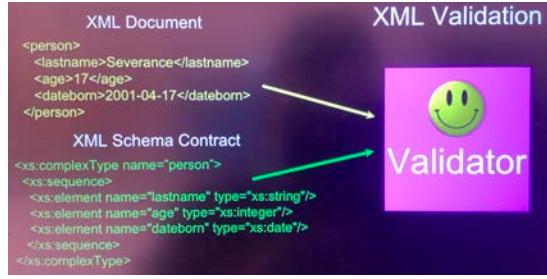
Tags: Beginning & ending of elements
Attributes: keyword/value pairs on the opening tag of XML
Serialize/De-Serialize - Convert data in one program into a common format can be stored and/or transmitted between systems in a programming language - independent manner

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



③ XML Schema → a language

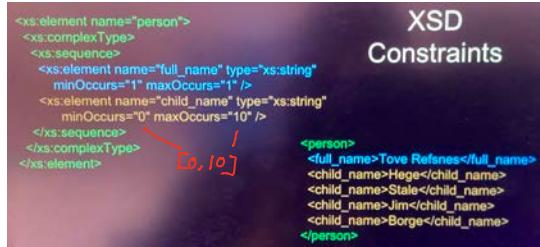
description of the legal format of an XML document
often used to specify a "contract" between systems - "My system will only accept XML that conforms to this particular Schema"
If a particular XML meets the specification of the Schema - it's said to be "validate"



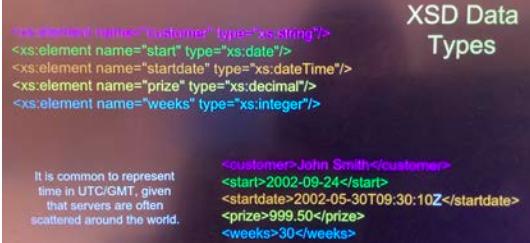
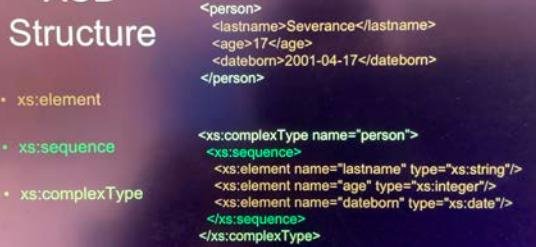
Many XML Schema Languages

- Document Type Definition (DTD)
- http://en.wikipedia.org/wiki/Document_Type_Definition
- Standard Generalized Markup Language (ISO 8879:1986 SGML)
- <http://en.wikipedia.org/wiki/SGML>
- XML Schema from W3C - (XSD)
- [http://en.wikipedia.org/wiki/XML_Schema_\(W3C\)](http://en.wikipedia.org/wiki/XML_Schema_(W3C))

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



XSD Structure



ISO 8601 Date/Time Format

2002-05-30T09:30:10Z

/ . ↑
Year-month-day Time of day Timezone - typically specified in UTC / GMT rather than local time zone.

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>'''

```

a list =>

```

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

```

```

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

```

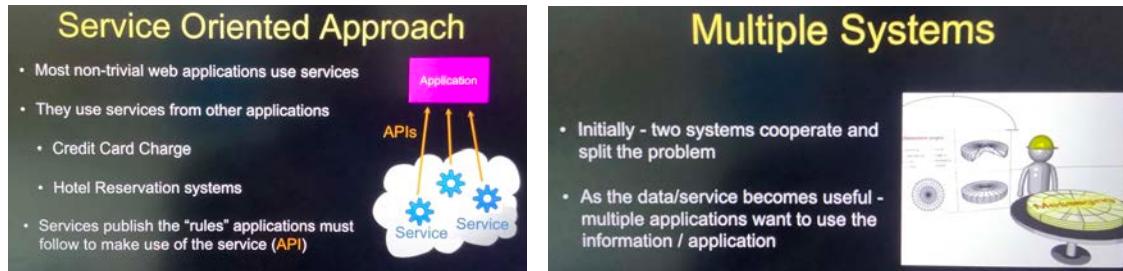
import json
data = '''
    "name" : "Chuck",
    "phone" : {
        "type" : "intl",
        "number" : "+1 734 303 4456"
    },
    "email" : {
        "hide" : "yes"
    }
'''
info = json.loads(data)
print('Name:', info['name'])
print('Hide:', info['email']['hide'])

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'])

```

Note: JSON is NOT a global standard, it's invented by a person and people love it.
 It is simpler 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>

```

import urllib.request, urllib.parse, urllib.error
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

while True:
    address = input('Enter location: ')
    if len(address) < 1: break

    url = serviceurl + urllib.parse.urlencode({'address': address})

    print('Retrieving', url)
    uh = urllib.request.urlopen(url)
    data = uh.read()
    print('Retrieved', len(data), 'characters')
    print(data[:20])
    js = None

    if not data or 'status' not in js or js['status'] != 'OK':
        print('==== Failure To Retrieve ====')
        print(data)
        continue

    lat = js["results"][0]["geometry"]["location"]["lat"]
    lon = js["results"][0]["geometry"]["location"]["lon"]
    print(lat, lon)
    location = js['results'][0]['formatted_address']
    print(location)

```

geojson.py

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

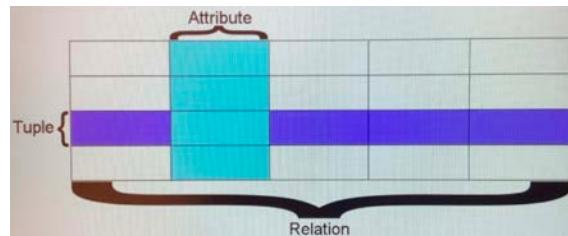
Lesson 15 – Object-Oriented Programming (terminology lecture)

Lesson 16 - Databases

Part I - 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)

- Database - contains many tables
 - Relation (or table) - contains tuples and attributes
 - Tuple (or row) - a set of fields that generally represents an "object" like a person or a music track
 - Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row

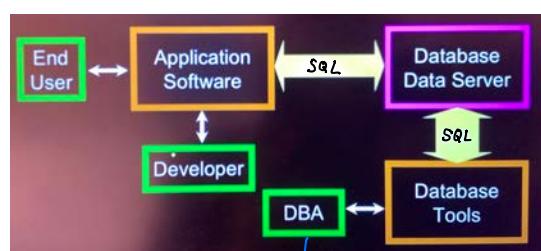
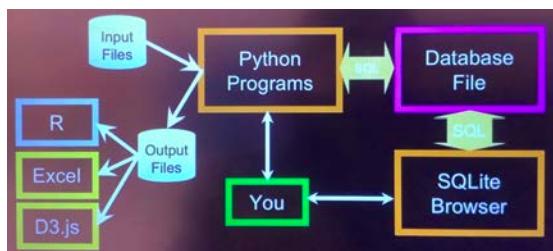


- ⑤ **Structured Query Language (SQL)** → the language we use to issue commands to the database

 - create a table
 - retrieve some data
 - insert data
 - delete data

(CRUD)

- ## ⑥ Data analysis application flow:



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

④ 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}, ...);`
 - (Insert)
 - (can be lowercase)
 - (value 1), value 2, ...)
 - ;
 - ← the semicolon important,
 - 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 lowercase)
 - (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 lowercase)
 - (can repeat)
 - execute in "Execute SQL"
- "Execute SQL" → <Codes line>: `SELECT * FROM {table name} WHERE {column name} = "value";`
 - (Retrieve → Select)
 - (can be lowercase)
 - (can repeat)
 - execute in "Execute SQL"
- "Execute SQL" → <Codes line>: `SELECT * FROM {table name} ORDER BY {column name} DESC;`
 - all columns
 - ;
 - 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', 'cv@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Sally', 'alb@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted', 'ted@umich.edu')
INSERT INTO Users (name, email) VALUES ('Kristen', 'kif@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.

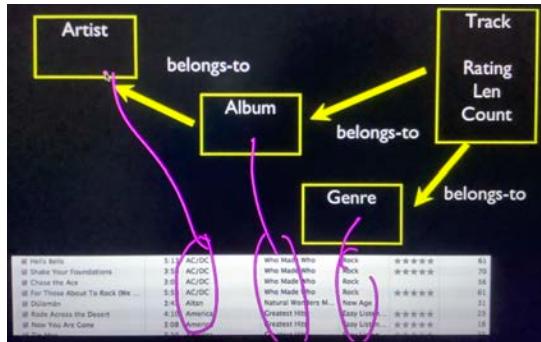
③ Project example:

Track	Len	Artist	Album	Genre	Rating	Count
0 Hell's Bells	5:13	AC/DC	Who Made Who	Rock	*****	61
1 Shake Your Foundations	3:14	AC/DC	Who Made Who	Rock	*****	70
2 Chase the Ace	3:01	AC/DC	Who Made Who	Rock	*****	56
3 For Those About To Rock (We	5:54	AC/DC	Who Made Who	Rock	*****	61
4 Distant Voices	3:45	Altan	Natural Wonders M.	New Age	*****	31
5 Road Across the Desert	4:10	America	Greatest Hits	Easy Listen...	*****	23
6 Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	*****	18
7 Tin Man	3:20	America	Greatest Hits	Easy Listen...	*****	23
8 Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	*****	24
9 Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	*****	26
10 Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	*****	18
11 Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	*****	22
12 Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	*****	14
13 Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	*****	21
14 War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	*****	21
15 Paranoid	2:53	Black Sabbath	Paranoid	Metal	*****	22
16 Planet Caravan	4:31	Black Sabbath	Paranoid	Metal	*****	25
17 Iron Man	5:59	Black Sabbath	Paranoid	Metal	*****	21
18 Doctor Funeral	4:53	Black Sabbath	Paranoid	Metal	*****	22
19 Hand of Doom	7:10	Black Sabbath	Paranoid	Metal	*****	23
20 Rat Salad	2:30	Black Sabbath	Paranoid	Metal	*****	31
21 Jack the Stripper/Fairies Wear	6:14	Black Sabbath	Paranoid	Metal	*****	24
22 Burnt Squad (TECH)	3:22	Brent	Brent's Album		1	
23 Guitars	4:36	Brent	Brent's Album		1	
24 Head	3:08	Brent	Brent's Album		1	
25 Hi metal man	4:20	Brent	Brent's Album		1	
26 Misters	2:58	Brent	Brent's Album		1	

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.



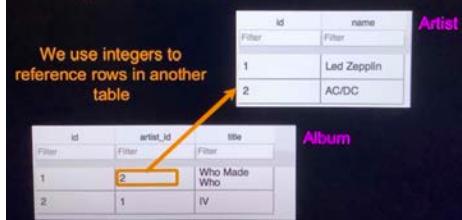
Part 4 - Represent relationships in a database

① Database Normalization (3NF)

- There is *tons* of database theory - way too much to understand without excessive predicate calculus
- Do not replicate data - reference data - point at data
- Use integers for keys and for references
- Add a special "key" column to each table which we will make references to. By convention, many programmers call this column "id"

http://en.wikipedia.org/wiki/Database_normalization

Integer Reference Pattern



② Three kinds of keys

- Primary key - generally an integer auto-increment field
- Logical key - What the outside world uses for lookup
- Foreign key - generally an integer key pointing to a row in another table



Key Rules

Best practices

- Never use your logical key as the primary key
- Logical keys can and do change, albeit slowly
- Relationships that are based on matching string fields are less efficient than integers

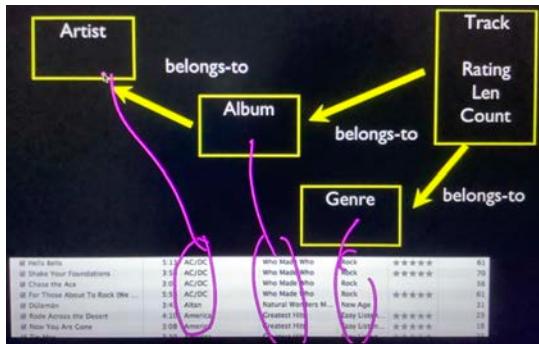
User
id
login
password
name
email
created_at
modified_at
login_at



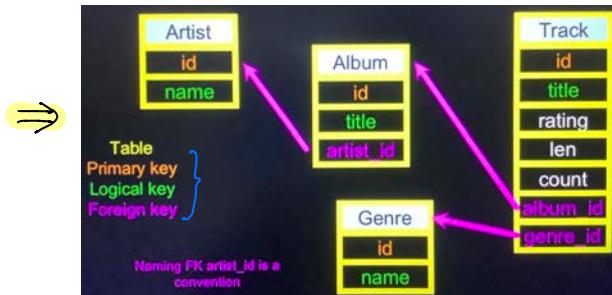
integer primary keys & foreign keys are very good.

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

order of construction: Artist & Genre | Album | Track
"JOIN"
(see next part)

```
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, 297, 0, 1, 2);  
  
SELECT Album.title, Artist.name FROM Album JOIN Artist ON Album.artist_id = Artist.id; Album → Artist hide key  
SELECT Album.title, Album.artist_id, Artist.id, Artist.name FROM Album JOIN Artist ON Album.artist_id = Artist.id; Album → Artist with keys  
SELECT Track.title, Track.genre_id, Genre.id, Genre.name FROM Track JOIN Genre ON Track.genre_id = Genre.id; Track → Genre hide keys  
SELECT Track.title, Artist.name, Album.title, Genre.name FROM Track JOIN Genre ON Track.genre_id = Genre.id AND Track.album_id = Album.id AND Album.artist_id = Artist.id; Track → all other tables
```

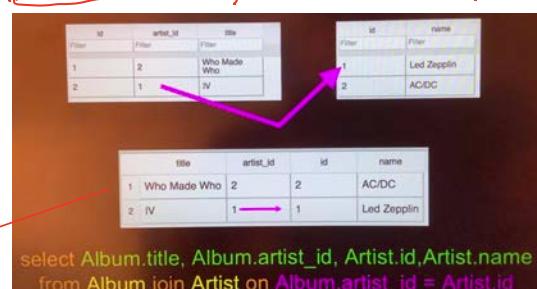
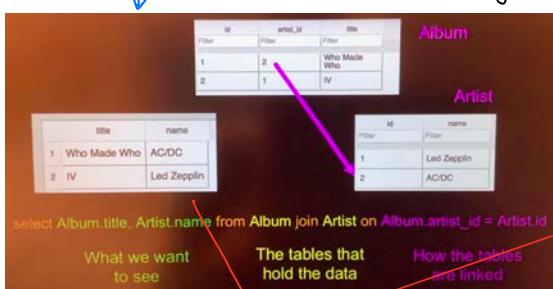
These are link tables we want to output

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

Syntax ↴



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

	title	name
1	Black Dog	Rock
2	Stairway	Rock
3	About to Rock	Metal
4	Who Made Who	Metal

id	title	genre_id	album_id	year	len	rating	count
1	Black Dog	1	207	6	0	0	0
2	Stairway	2	482	6	0	0	0
3	About to Rock	2	313	6	0	0	0
4	Who Made Who	1	207	6	0	0	0

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

What we want to see
The tables that hold the data
How the tables are linked

What we want to see
The tables which hold the data
How the tables are linked

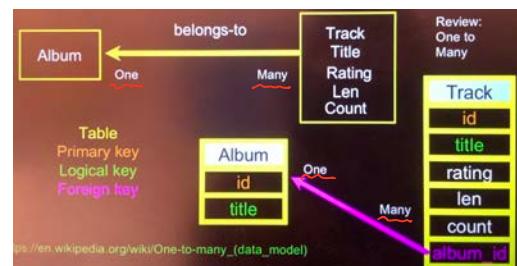
② **ON Clause is Important:** If no "ON" clause → will give all possible combinations of rows & cause mismatch

SELECT Track.title,
Track.genre_id,
Genre.id, Genre.name
FROM Track JOIN Genre
This is NOT good.

Joining two tables without an
ON clause gives all possible
combinations of rows.

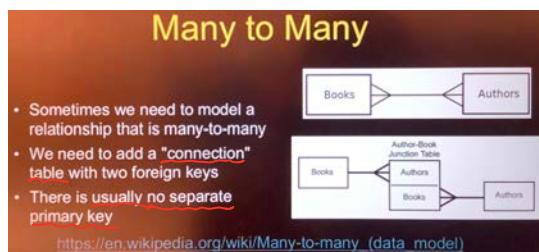
title	genre_id	id	name
1 Black Dog	1	1	Rock
2 Black Dog	1	2	Metal
3 Stairway	1	1	Rock
4 Stairway	1	2	Metal
5 About to Rock	2	1	Rock
6 About to Rock	2	2	Metal
7 Who Made Who	2	1	Rock
8 Who Made Who	2	2	Metal

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**



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

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

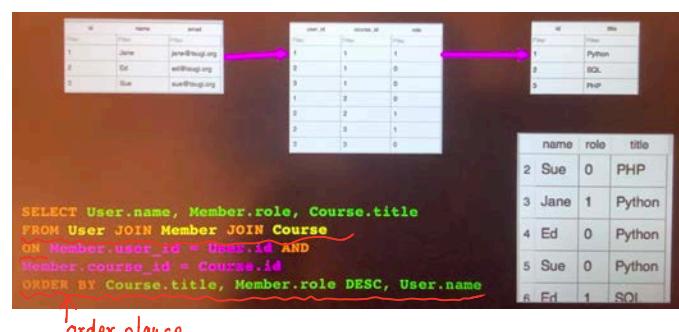
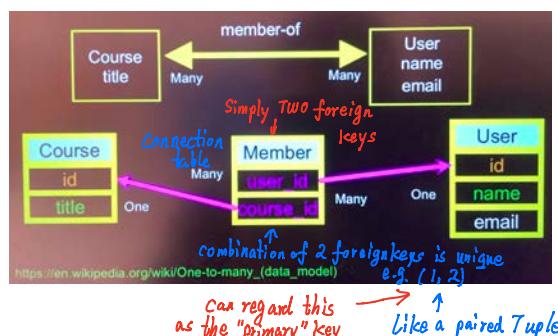
INSERT INTO Member (user_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 1, 0);

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
    
```

③ Project as an example



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"

```
terminal> libscay -> tracks.py
tracks> tracks.py
import xml.etree.ElementTree as ET
import sqlite3
[ The file NOT necessarily
conn = sqlite3.connect('trackdb.sqlite')          exists
cur = conn.cursor()

# Make some fresh tables using executeScript()
cur.execute('''
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
);
[ Constraint-blown away
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
);
''')

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

# ckeyTrack ID</key><integer>369</integer>
# ckeyName</key><string>Another One Bites The Dust</string>
# ckeyArtist</key><string>Queen</string>
def lookup(d, key):
    found = False
    for child in d:
        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))
```

```
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) avoid
    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() ← 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'

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

```
for entry in json_data:
    name = entry[0];
    title = entry[1];

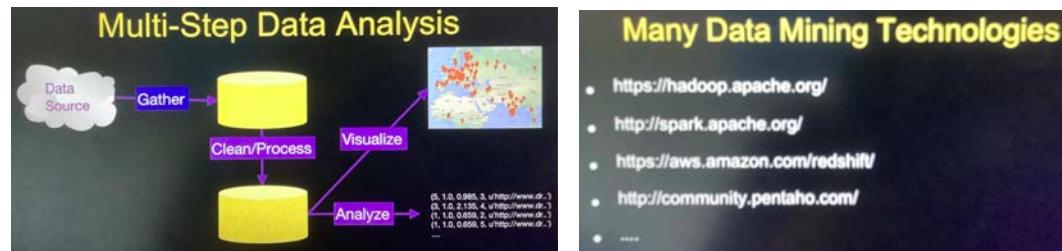
    print((name, title))

    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

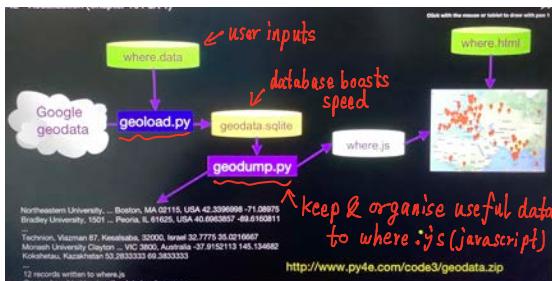


Many Data Mining Technologies

- <https://hadoop.apache.org/>
- <http://spark.apache.org/>
- <https://aws.amazon.com/redshift/>
- <http://community.pentaho.com/>
-

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

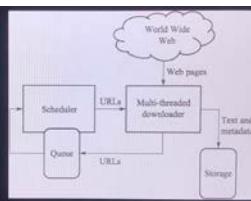
- (Google Rank)
- write a simple web page crawler
- compute a simple version of Google Page Rank Algorithm
- visualize the resulting network

- Web crawling
- Index building
- Searching

①

Web Crawler

- Retrieve a page
- Look through the page for links
- Add the links to a list of "to be retrieved" sites
- Repeat...



http://en.wikipedia.org/wiki/Web_crawler

Web Crawling Policy

- a **selection policy** that states which pages to download,
- a **re-visit policy** that states when to check for changes to the pages,
- a **politeness policy** that states how to avoid overloading Web sites, and
- a **parallelization policy** that states how to coordinate distributed Web crawlers

②

robots.txt

- A way for a web site to communicate with web crawlers
 - An informal and voluntary standard
 - Sometimes folks make a "Spider Trap" to catch "bad" spiders
- http://en.wikipedia.org/wiki/Robots_Exclusion_Standard
http://en.wikipedia.org/wiki/Spider_trap

User-agent: *
Disallow: /cgi-bin/
Disallow: /images/
Disallow: /tmp/
Disallow: /private/

Index Building

