__/__/____

For eg → email → spam
→ not spam.

A classification

⟹ Python Data Structures
→ String
→ List
→ Tuple
→ Dictionary

String

```
>>> stuff = 'Hello World'
>>> type (stuff)
<class 'str'>
```
⟶ type of datastructure

```
>>> dir (stuff)
['capitalise', 'casefold', 'center' - - - - - ]
```
⟶ All the available methods of data structure

NOTE → In dir() ignore the ones with underscores they are used by python although hidden is still counted as character.

Files

```
>>> stuff = 'X\nY'
>>> print (stuff)
X
Y
>>> len (stuff)
3
```

```
>>> fhand = open ('mbox - short.txt').
>>> inp = fhand.read()
>>> print (len(inp))
94626
```

• read() file ko lambi string mein convert kardeta hai

__/__/____

Dictionary

```
counts = dict()
names = ['a', 'b', 'c', 'd']
for name in names:
    if name not in counts
    counts[name] = 1
    else:
        counts[name] += 1
```

```
counts[name] = counts.get(name, 0) + 1
```

.get() returns the value to key (name) if it doesn't exits it returns default (0)

In this case

Tuples

dict.items() → Return List of tuples of key value pair

Eg → dict = ['a': 2, 'b': 5, 'c': 1, 'd': 10]

```
list = [] or list()
for key, val in dict.items()
    new tup = (val, key)
    list.append(new tup)
```

→ Value first then key

```
list = sorted(list, reverse = True)
```

→ Sort based on value.

__/__/__

Short version — imagine as 'forall'

```
>>> print ( sorted ( [ (v,k) for k,v in dict.items() ] ) )
```

one line.

⇒ Tuples are comparable

```
>>> (0,+,2) < (5,+,2)
True
```
→ checks first ignores rest ; i.e 0 < 5 True.)

```
>>> (0,1,2000) < (0,3,+)
True
```
→ checks first 0 : 0 now second 1 < 3 → True

```
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
```

⇒ PYTHON TO ACCESS WEB DATA.

→ Regular Expressions / Regexp

^ → matches beginning of line

$ → matches the end of line

. → matches any character also called WILD CARD CHARACTER

\s → matches whitespace

\S → matches any non-whitespace character

* → Repeates matches character zero or more time

*? → same as * but non-greedy

+ → Repeates a character one or more time

+? → same as + but non-greedy

__/__/____

[aeiou] → Matches single character in listed set

[^XYZ] → Matches single character not in listed set

[a-z 0-9] → The set of characters can include range

(→ Indicates where string extraction is to start

) → " " " " " " end

Library → import re

re.search() → to see if string matches a regexp.

re.findall () → extract portion of string that matches regexp.

Eg →

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

← In index is greater than 0

```
if re.search('From:', line):
    print(line)
```

→ It returns True/False.

if line.startswith('From:'): → if re.search('^From:', line)

Regexp used is [0-9]+ (any digit, one or more times)

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

* and + uses GREED Y MATCHING i.e. the push outward and find largest possible string

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

First F, one or more chars, last character

and not ['From:'] because its greedy.

```
>>> y = re.findall('^F.+?:', x)
>>> print(y)
['From:']
```

not greedy.

'@([^]*)' → Eg abc@d.com

one or more. space. match non blank characters

↳ d.com gets extracted because of ()

$ → It stands for something in regexp but \$ → It means real dollar sign.

__/__/____

Socket = It is an endpoint of a bi-directional inter-process communication flow across an Internet Protocol based computer network, such as the internet.

Using in python

just make Connection

```
import socket
mysock = socket.socket(socket.AF_INET,
                       socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
```

'data.pr4e.org' ← Host name
80 ← Port

```
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0
      \n\n'.encode()
```

depicts 2 enter if error replace with \r\n\r\n

```
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break          ← If no data in file
    print(data.decode())
mysock.close
```

>>> print (ord ('H')) ord() helps with ASCII
72

· encode () → String to Bytes

Since HTTP is common, we have library that does all socket work.

```
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())
```
↳ to remove extra \n as print puts \n for next iteration of loop

NOTE → It gives only data not header.
For headers → headers = dict(fhand.getheaders())

PARSING WEB PAGES / WEB SCRAPPING.

→ Intro to Beautiful Soup

```
Import BeautifulSoup
url = input()
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')
# Retrieve all anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

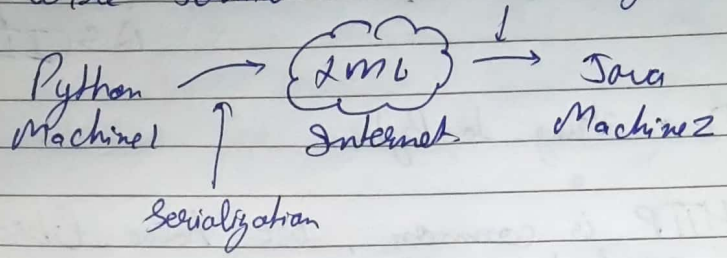
soup ← soup object

read() → one String

__/__/____

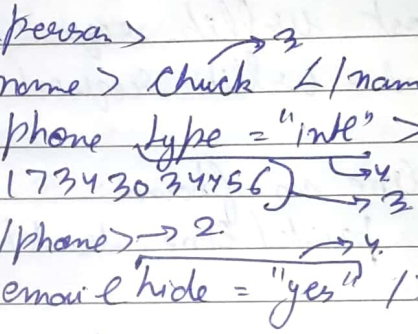
Data on Web.

→ Wire Format



XML

1. Start tag
2. End tag
3. Text content
4. Attribute
5. Self Closing tag.



Using in python

import ~~xml.tree.E~~ xml.etree.ElementTree as ET

data = '''<person>
multiline strings.
<name> Chuck </name>
:
</person>'''
end.

tree = ET.fromstring(data) → forms an XML traversal tree and if fails if bad XML in data like syntax error.

print(tree.find('name').text)
↓ tag name
↓ data

print(tree.find('email').get('hide'))
↓ tag name.
↓ attribute name.
returns yes or ~~no~~

stuff = ET.fromstring(input)

lst = stuff.findall('users/user') or (.//users)

↓ list of tags

↓ parent = users, child = user

↓ find out probably all user tag regardless of parent not sure

print (len(lst)) → total no. of users.

for item in lst:

    print (item.find('name').text) → name of all users one by one

XML → complex JSON → simple

↓ very powerful

JSON → Java Script Object Notation.

use in python

```
import json
data = '''{
    "name" : "Chuck",
    "phone": {
        "type" : "intl",
        "number" : "+1 234"
    }
    "email": {
        "hide" = "yes"
    }
}'''
```

''' when taking from url make sure to apply .read() first

```
info = json.loads(data)
print (info["name"]) / print (info["email"]["hide"])
```

print (len(info))

JSON represents data as nested "lists" and "dictionaries".

API → Application Programming Interface

→ API Security and Rate Limiting.

Google map free limit till → 2500 req. per day

agar json ke output ke aage b' laga ho it means it is byte array and JSON is not decoded.

PYTHON & DATABASES

Object Oriented Programming

Classes + a - A template
Method or Message - A defined capability of class
Field or attribute - A bit of data in class
Object or Instance - a particular instance of class.

```
Class PA:
  x = 0
  def party(self):
    self.x = self.x + 1
    print(self.x)

an = PA()
an.party()
```

self.x means an.x because an is an object calling party.

__/__/____

Constructor def __init__(self)

Deconstructor def __del__(self)

Constructor is typically used to set up variables
Destructor is seldom used.

Inheritance.

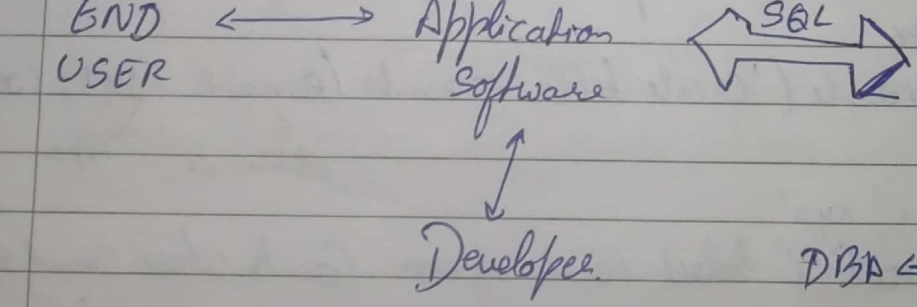
name of inherited class
↓
class A (B)

Relational Database

Database -
Relation (or table) - Set of tuples with same Attributes
Row (or tuple) - An object & its info.
Attribute (column or field) -

SQL - Structured Query Language

LARGE PROJECT STRUCTURE



Database Model/Schema → Structure of / Format of database.

~~Create Table~~ → CREATE TABLE Users (
name VARCHAR (128),
email VARCHAR (128)
)

In python cur.execute('...') → Execute one line
cur.executescript('...') → multiple SQL lines separated by ;

INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')

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

Note → without from all rows will be deleted.

UPDATE Users SET ~~no~~ name = 'ABL' WHERE email = 'zxy@com'

SELECT * FROM ~~USERS~~ Users

SELECT * FROM ~~USERS~~ Users WHERE email = "zxy@on"

Select * from Users order by name

Program

import sqlite3

Creates a new connection
conn = sqlite3.connect('emaildb.sqlite')
cur = conn.cursor()

Drop if already exist
cur.execute('Drop table if exists counts')
table name

~~cur.ex~~

Create new table
cur.execute('Create table counts (email TEXT, count INTEGER)')
attribute Type

email = 'xyz'
cur.execute('Select count from Counts where email = ?', (email,))

row = cur.fetchone()
If row is None:
else

conn.commit()

sqlstr = "Select email, count from Counts ORDER BY count DESC LIMIT 10";
Descending → Top 10.

for row in cur.execute(sqlstr):
    print (str(row[0]), row[1])

cur.close()

Rows returned are in form of tuples.

Primary Key → Main Key
Foreign Key → Key referring to Main Key of other table.
Logical Key → Might be looking up data using this key.

Create Table Album (
    id INTEGER NOT NULL AUTOINCREMENT
    UNIQUE PRIMARY KEY

JOIN →

table name / field name

Select Album.title, Artist.name from Album join Artist on Album.artist_id = Artist.id

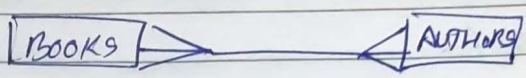
NOTE if we do not use ON clause it basically says all combinations of both tables

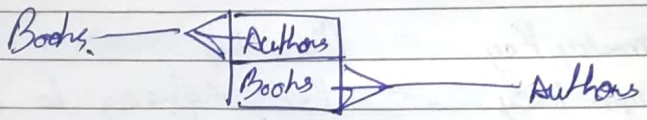
INSERT OR IGNORE → can be used when we have unique attribute. It will not ~~add~~ insert same attribute twice.

__/__/____

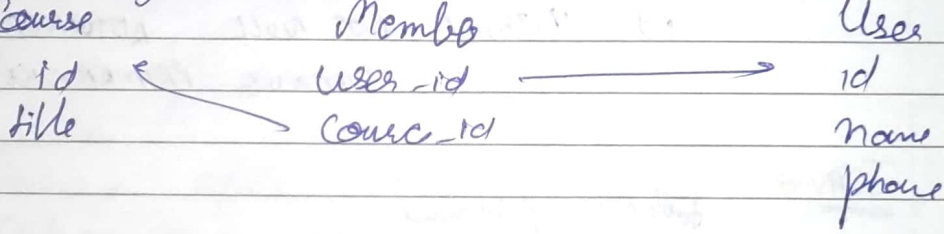
MANY TO MANY

We break the tables having MANY to MANY using a table called "connection" table with two foreign keys.





There is usually no primary key in such connection table but we ~~can~~ have multiple columns or all colums as primary key.



one user many courses
many ~~courses~~ users one course } Many to Many.

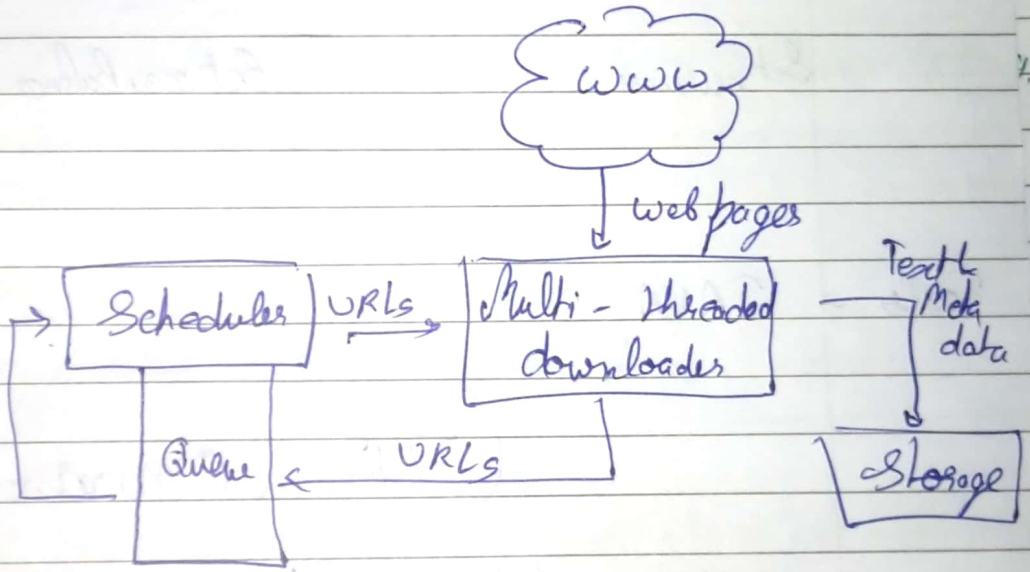
⇒ Retrieving Processing and Visualizing Data.

Search Engine Architecture → Web Crawling, Index Building, Searching.

Web Crawler → Computer program that browses WWW in methodical, automated manner, Creates a copy of all visited pages for later

processing by a search engine that will index the downloaded pages to provide fast searches.

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



INDEX BUILDING

→ Small project to crawl web store in database, page rank, visualization.