ST445 Managing and Visualizing Data

Using data from the Internet

Week 4 Lecture, MT 2017 - Kenneth Benoit, Dr. Akitaka Matsuo

Plan for today

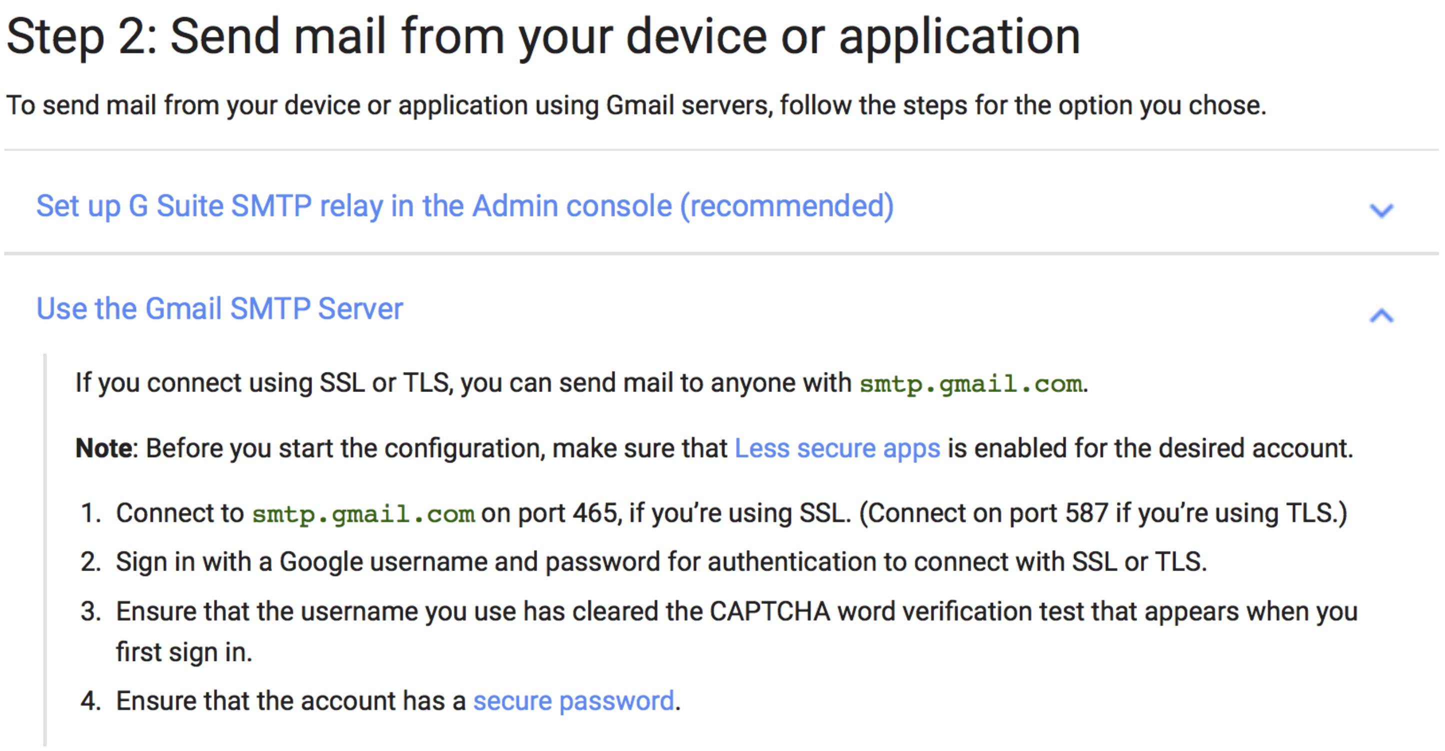
- the basics of TCP/IP and how the internet works
  - Network architecture
  - URLs/URIs
  - the client-server model
- HTML, XML, other data Markup languages
- Web scraping

Protocols and server requests

- Client requests a connection to a server
- This is done using a specific *protocol*

- Example: Email client sends a request to send a message using SMTP (Simple Mail Transfer Protocol)

  For Google: (https://support.google.com/a/answer/176600?hl=en)



Step 2: Send mail from your device or application

To send mail from your device or application using Gmail servers, follow the steps for the option you chose.

Set up G Suite SMTP relay in the Admin console (recommended)

Use the Gmail SMTP Server

If you connect using SSL or TLS, you can send mail to anyone with `smtp.gmail.com`.

Note: Before you start the configuration, make sure that Less secure apps is enabled for the desired account.

1. Connect to `smtp.gmail.com` on port 465, if you're using SSL. (Connect on port 587 if you're using TLS.)
2. Sign in with a Google username and password for authentication to connect with SSL or TLS.
3. Ensure that the username you use has cleared the CAPTCHA word verification test that appears when you first sign in.
4. Ensure that the account has a secure password.

Network architecture

Layer	**Examples**
Application	DNS, TFTP, TLS/SSL, FTP, HTTP, IMAP4, POP3, SIP, SMTP, SNMP, SSH, Telnet, RTP
Transport	TCP (https://en.wikipedia.org/wiki/Transmission_Control_Protocol), UDP (https://en.wikipedia.org/wiki/User_Datagram_Protocol)
Internet	IP (IPv4, IPv6), ICMP (https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol), IGMP (https://en.wikipedia.org/wiki/Internet_Group_Management_Protocol)
Link	ARP (https://en.wikipedia.org/wiki/Address_Resolution_Protocol)

Link Layer

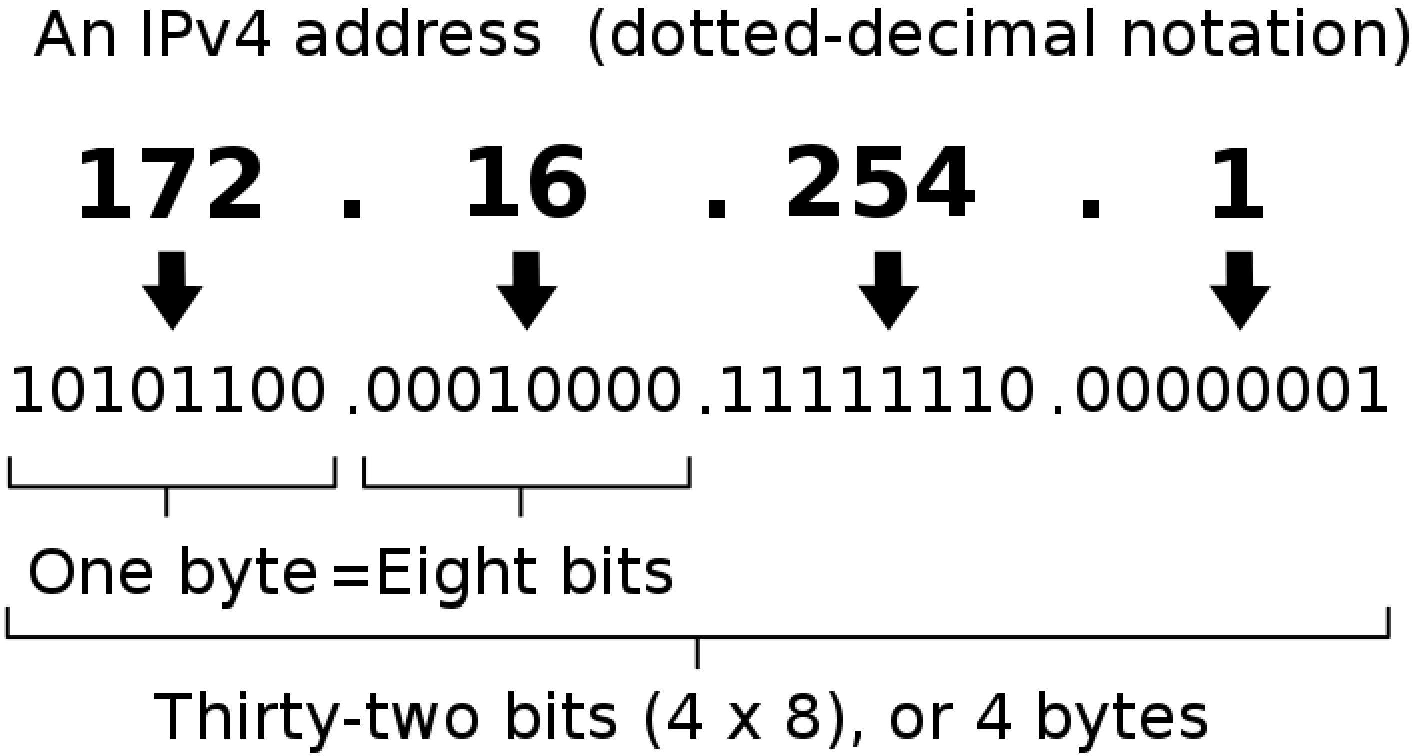
- Connects the computer to its local network, and moves data from server to server (in "hops")
    - Example: your laptop's Wifi connection to a base station
- Needs to know how to *encode* and *send* data across the link
- Needs to manage multiple computers sending data at the same time
- Breaks data into *packets* and sends each separately

Internet(work) Layer (IP)

- "Routers" are like traffic cops directing packet traffic toward their destinations, to get packets to the computers where they are aimed
- Routers will redirect packets to get as close as possible, and early steps are only approximate
- This is why the Internet is so robust, since there are many routes to the same destination

- Data is sent in *datagrams*, consisting of a *header* and a *payload*
  - The IP **header** includes source IP address, destination IP address, and other metadata needed to route and deliver the datagram
  - The **payload** is the data that is transported
  - The method of nesting the data payload in a packet with a header is called **encapsulation**
- Examples:
  - IPv4
  - IPv6

IPv4



An IPv4 address (dotted-decimal notation)

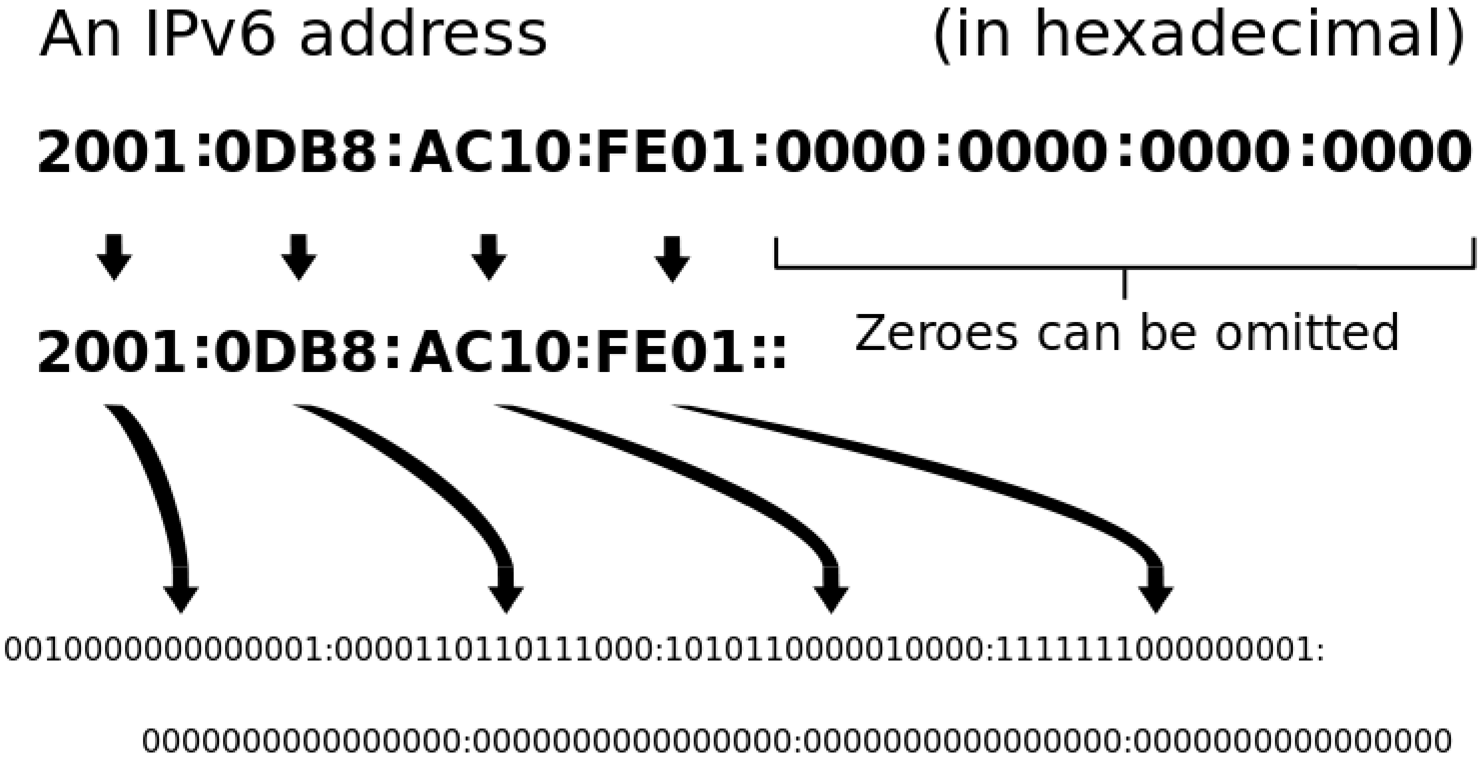
172 . 16 . 254 . 1

10101100 .00010000 .11111110 .00000001

One byte=Eight bits

Thirty-two bits (4 x 8), or 4 bytes

IPv6



An IPv6 address (in hexadecimal)

2001:0DB8:AC10:FE01:0000:0000:0000:0000

2001:0DB8:AC10:FE01::

Zeroes can be omitted

0010000000000001:0000110110111000:1010110000010000:1111111000000001:

0000000000000000:0000000000000000:0000000000000000:0000000000000000

Transport Layer

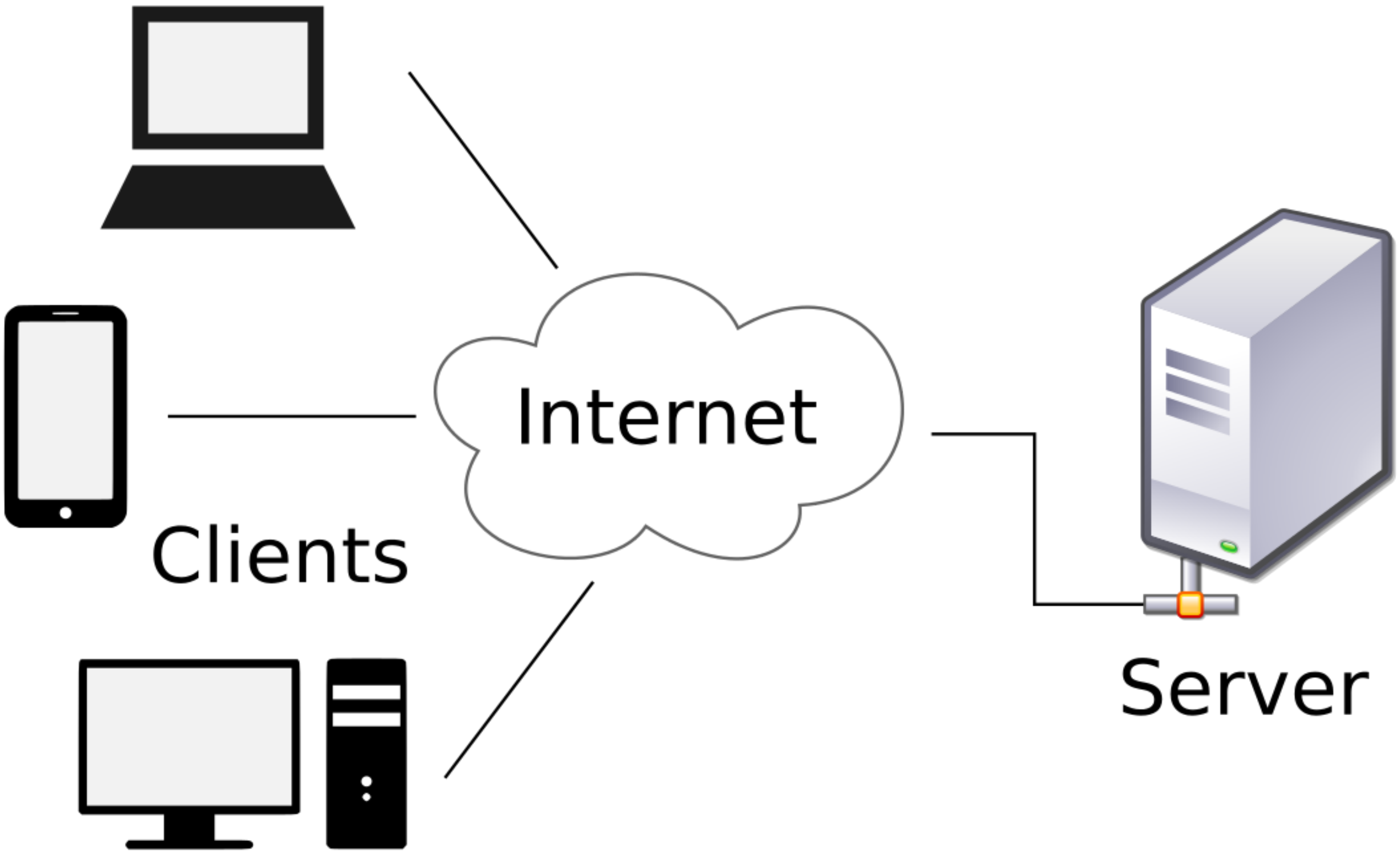
- Describes how the destination computer can reconstruct the original data, even if packets were received out of order
- Allows for variable "window sizes" describing the amount of data that source computer will spend waiting for acknowledgement, before sending more data
- This allows for the same transport protocol across both fast and slow network connections

Application Layer

- This is where the **client-server model** comes in: applications are split between a (destination) server computer and a (source) client computer
- These require an "application protocol", such as http, ftp, smtp, etc.

Client-server model

- Client: user computer; tablet; phone; software application; etc.
- Server: Jupyter server on Fabian; mail server; file server; web server; etc.



Network architecture (revisited)

Layer	Examples
Application	DNS, TFTP, TLS/SSL, FTP, HTTP, IMAP4, POP3, SIP, SMTP, SNMP, SSH, Telnet, RTP
Transport	TCP (https://en.wikipedia.org/wiki/Transmission_Control_Protocol), UDP (https://en.wikipedia.org/wiki/User_Datagram_Protocol)
Internet	IP (IPv4, IPv6), ICMP (https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol), IGMP (https://en.wikipedia.org/wiki/Internet_Group_Management_Protocol)
Link	ARP (https://en.wikipedia.org/wiki/Address_Resolution_Protocol)

Uniform Resource Locators (URLs)

- A reference to a web resource, used by the application layer
- Syntax:

```
 scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

- Examples:

```
 http://localhost:8888/notebooks/GitHub/lse-st445/lectures/week04/ST445_
wk4_lecture.ipynb
 https://en.wikipedia.org/wiki/URL
 ftp://example.myftpserver.com/files/myfiles
```

Internationalized URLs: What about `http://例子.卷筒纸`??

- Solution: Internationalized Resource Identifier (IRI), a form of URL that includes Unicode characters
    - Automatic converson of the domain name into punycode (https://en.wikipedia.org/wiki/Punycode) usable by the Domain Name System (https://en.wikipedia.org/wiki/Domain_Name_System)
    - any characters not part of the basic URL character set are escaped as hexadecimal using percent-encoding (https://en.wikipedia.org/wiki/Percent-encoding)

- Examples:

    - `http://例子.卷筒纸`becomes `http://xn--fsqu00a.xn--3lr804guic/`
    - `http://example.com/引き割り.html` becomes `http://example.com/%E5%BC%95%E3%81%8D%E5%89%B2%E3%82%8A.html`

HTML

- HTML = Hyper Text Markup Language
- HTML is the standard markup language to create webpages
- HTML consists of a various kind of tags
    - `html`
    - `head` and `body`
    - `h1`, `h2`, `h3`...
    - `p` (paragraph)
    - `a` link tag (e.g. hyperlink in the text)
    - `img`

- tags opens with `<tagname>` and closes with `</tagname>`
- tags have attributes such as:
  - `id`
  - `class`
  - `href`

HTML

A simplest html file

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro (https://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro)

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

HTML

Another simple html file

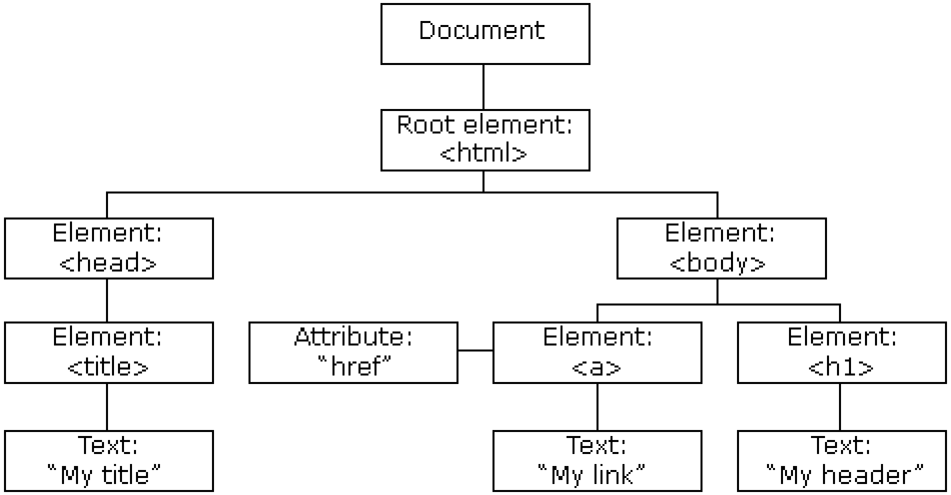
```
<!DOCTYPE html>
<html>
<head>
<title>My Title</title>
</head>
<body>

<h1>My Header</h1>
<a href="http://kenbenoit.net">My link</a>

</body>
</html>
```

HTML

The HTML in the previous page can be presented in a tree like this:



HTML

- HTML displays mostly presents **static** contents.
- Many contents of dynamic webpages cannot be found anywhere in html
    - Example: google maps
- Understanding what's static and what's dynamic in a webpage is a crucial first step for web scraping

Other data formats: XML

- XML = eXtensible Markup Language
- XML is used for distributing data over the Internet.
    - Examples:
        - RSS (web feeds): http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907 (http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907)
        - SVG (graphic): https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg (https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg)
        - epub (books)
        - Office documents (OpenOffice, MS)
- XML looks a lot like HTML, but more frexible (e.g. basically no preset definitions of tags).

XML, Example 1 (no schema)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notes>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
<note>
  <to>Jason</to>
  <from>Kelly</from>
  <heading>Offer</heading>
  <body>You won 10M. Contact us immediately.</body>
</note>
</notes>
```

- This file contains two notes, seems to have common strcuture for notes but you never know!

XML, Example 2 (with DTD)

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
</note>
```

- This XML has a DTD (Document Type Definition)
- DTD is one of the XML schematic languages, that are used as a validator of data input

Other data formats: JSON

- JSON = **J**ava**S**cript **O**bject **N**otation
- Another format for data exchange in the net
- Lightweight, easy to read, less formatted
- Written with JavaScript object notation, but independent from any language
- Used in many APIs including (See here (https://www.sitepoint.com/10-example-json-files/)):
    - Twitter
    - Facebook
    - YouTube

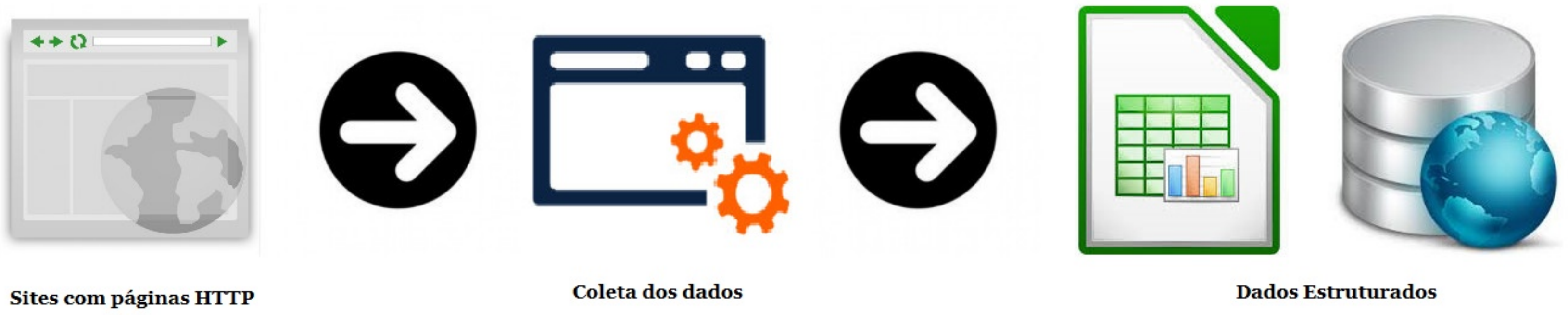
JSON Example

```
{
    "note" : {
        "to" : "Tove",
        "from" : "Jani",
        "heading" : "Reminder",
        "body" : "Don't forget me this weekend!"
    },
    "note" : {
        "to" : "Jason",
        "from" : "Kelly",
        "heading" : "Offer",
        "body" : "You won 10M. Contact us immediately."
    }
}
```

Web scraping

What is it?

"Web scraping (web harvesting or web data extraction) is data scraping used for extracting data from websites" [Wikipedia: Web Scraping (https://en.wikipedia.org/wiki/Web_scraping)](https://en.wikipedia.org/wiki/Web_scraping)



Web-scraping steps

1. Get contents from the web
2. Extract information
3. Reshape and save the information as data

Get contents from the web

- First of all you need to know where is the information
- Examples:
  - Government's administrative data
  - Newspaper websites
- The data format
  - web-pages (in html)
  - data files in various format (csv, spss, stata)
  - document files (MS-Word, pdf)
  - API (e.g. JSON)
  - pictures

Get to know the target website

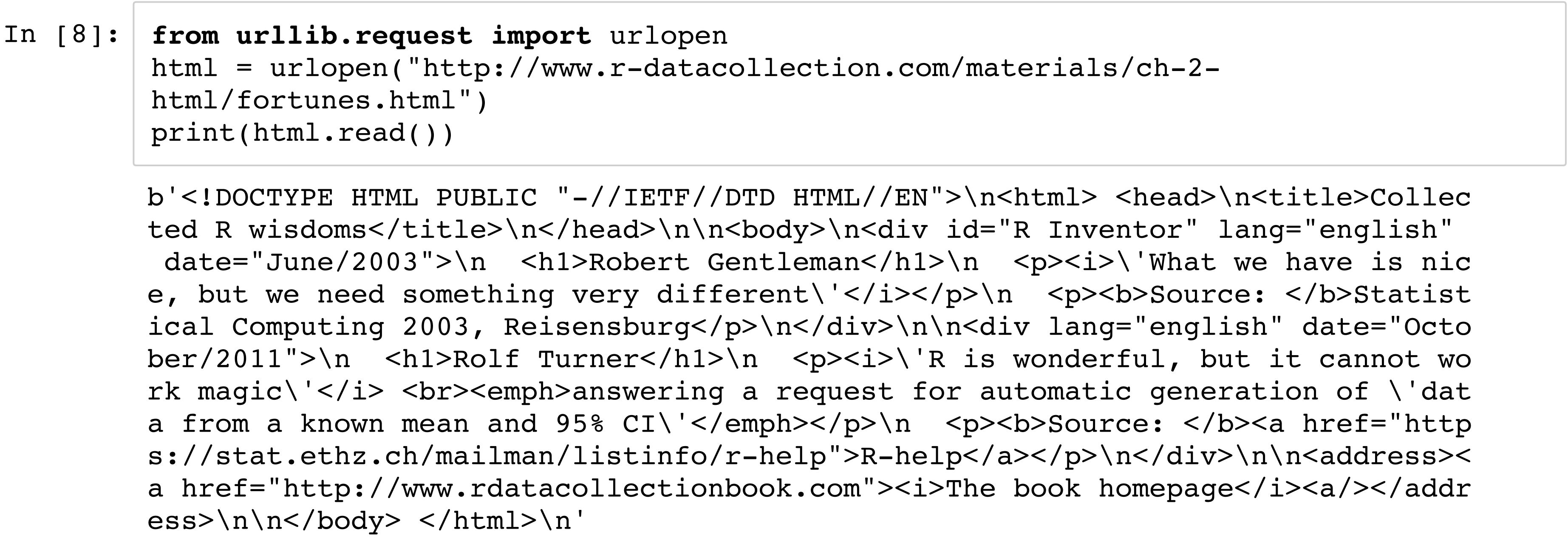
1. Open the website, learn how it's structured
2. "View page source" and "Inspect"
    - Example 0 (http://www.r-datacollection.com/materials/ch-2-html/fortunes.html)
    - Example 1 (http://www.r-datacollection.com/materials/ch-6-ajax/fortunes/fortunes1.html)
    - Example 2 (http://www.r-datacollection.com/materials/ch-6-ajax/fortunes/fortunes2.html)
    - Example 3 (http://www.r-datacollection.com/materials/ch-6-ajax/fortunes/fortunes3.html)

These examples looks similar (especially Ex 0 and Ex 2) but the static contents are different, so what a normal scraper can see might be different.

Get webpage contents

- Suppose that I you know that what you want to get is in static contents of the webpage (i.e. something you can find in "View page source")
- Then steps are
    1. Get the page contents
    2. Parse the contents
    3. Extract and format the contents

Get webpage contents in Python



In [8]:

```
from urllib.request import urlopen
html = urlopen("http://www.r-datacollection.com/materials/ch-2-
html/fortunes.html")
print(html.read())
```

```
b'<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">\n<html> <head>\n<title>Collec
ted R wisdoms</title>\n</head>\n\n<body>\n<div id="R Inventor" lang="english"
 date="June/2003">\n  <h1>Robert Gentleman</h1>\n  <p><i>\'What we have is nic
e, but we need something very different\'</i></p>\n  <p><b>Source: </b>Statist
ical Computing 2003, Reisensburg</p>\n</div>\n\n<div lang="english" date="Octo
ber/2011">\n  <h1>Rolf Turner</h1>\n  <p><i>\'R is wonderful, but it cannot wo
rk magic\'</i> <br><emph>answering a request for automatic generation of \'dat
a from a known mean and 95% CI\'</emph></p>\n  <p><b>Source: </b><a href="http
s://stat.ethz.ch/mailman/listinfo/r-help">R-help</a></p>\n</div>\n\n<address><
a href="http://www.rdatacollectionbook.com"><i>The book homepage</i><a/></addr
ess>\n\n</body> </html>\n'
```

Get webpage contents in R

```
url <- "http://www.r-datacollection.com/materials/html/fortunes.html"
fortunes <- readLines(con = url)
cat(fortunes)
## <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN"> <html> <head> <title>Collected
 R wisdoms</title> </head>  <body> <div id="R Inventor" lang="english" date="Jun
e/2003">   <h1>Robert Gentleman</h1>   <p><i>'What we have is nice, but we need s
omething very different'</i></p>   <p><b>Source: </b>Statistical Computing 2003,
 Reisensburg </div>  <div lang=english date="October/2011">   <h1>Rolf Turner</h1
>   <p><i>'R is wonderful, but it cannot work magic'</i> <br><emph>answering a re
quest for automatic generation of 'data from a known mean and 95% CI'</emph></p>
  <p><b>Source: </b><a href="https://stat.ethz.ch/mailman/listinfo/r-help">R-help
</a></p> </div>  <address><a href="www.r-datacollectionbook.com"><i>The book home
page</i><a/></address>  </body> </html>
```

HTML Parsing

The next step is to parse the content of html

A very simple example

```
In [9]: from bs4 import BeautifulSoup

        html = urlopen("http://www.r-datacollection.com/materials/ch-2-
        html/fortunes.html")
        bsObj = BeautifulSoup(html, "html.parser")
        nameList = bsObj.findAll("h1") # this line extract "h1" tags
        for name in nameList: # this loop print out the content of "h1" tags
            print(name.get_text())
```

```
Robert Gentleman
Rolf Turner
```

XPath

You may want to navigate through html structure to get a particular information.

Example

- Select in the text of `<i>`-tag inside `<p>`-tag
- Select based on the class value (this can be achieved with BeautifulSoup, though)

Use `etree` in `lxml`

In [3]:

```
from lxml import etree

parser = etree.HTMLParser()
tree = etree.parse("http://www.r-datacollection.com/materials/ch-2-html/fortunes.html", parser)
h1nodes = tree.xpath('.//div/h1') # find the h1 in div
for nod in h1nodes:
    print(nod.text)
```

```
Robert Gentleman
Rolf Turner
```

In [4]:

```
h1nodes_oct2011 = tree.xpath('.//div[@date="October/2011"]/h1') # find the h1 in div with specific date value
for nod in h1nodes_oct2011:
    print(nod.text)
```

```
Rolf Turner
```

R web scraping toolbox

- Get contents
    - `RCurl`
    - `httr`
- Parse and extract information
    - parsing and analyzing markup language:
        - `XML`
        - `XML2`
    - content extraction with matching
        - (base R)
        - `stringr`
        - `sgringi`

Python web scraping toolbox

- Get contents
    - `urllib`
    - `httplib`
    - `requests`
- Parse and extract information
    - parsing and analyzing markup language:
        - `bs4` (`BeautifulSoup`)
        - `lxml`
    - content extraction with matching
        - `re`

Selenium

- Standard tools for web scraping (e.g. `httr` in R or `urllib` in Python) may not work in some occasions
- Reasons:
  - "Some websites don’t like to be webscraped. In these cases you may need to disguise your webscraping bot as a human being. Selenium is just the tool for that." webscraping with Selenium (http://thiagomarzagao.com/2013/11/12/webscraping-with-selenium-part-1/)
  - The information is in non-static contents
- Solution:
  - Use `selenium` = an automated testing suite for web applications
  - Manipulate actual web-browser (e.g. Chrome, Firefox) using selenium drivers

WIth selenium, you should be able to get whatever you can get with your browser (theoretically speaking...)

Caveats

Web-scraping is not always (or never) welcomed by site-owners

Why?

- excessive traffic
- influence on their revenues

You can be warned, blocked, and even sued.

So, what to do?

1. Read TOC carefully
2. Check `robot.txt` (c.f. http://www.robotstxt.org/ (http://www.robotstxt.org/))
3. Get permission if possible
4. Be nice
    - place short breaks between fetching
    - scrape during off-peak hours
    - avoid scraping exessive materials

Further reading

Python

- Automate the Boring Stuff with Python (https://automatetheboringstuff.com/)
- Web Scraping with Python (http://shop.oreilly.com/product/0636920034391.do)

R

- Automated Data Collection with R (http://www.r-datacollection.com/)

Coming soon

- **Lab**: Simple web-scraping exercise
- **Next week**: APIs, getting and analyzing Twitter data, working with text