WWW and Search

- World Wide Web
- Python WWW API
- String Pattern Matching
- Web Crawling

World Wide Web (WWW)

The Internet is a global network that connects computers around the world

- It allows two programs running on two computers to communicate
- The programs typically communicate using a client/server protocol: one program (the client) requests a resource from another (the server)
- The computer hosting the server program is often referred to as a server too

The World Wide Web (WWW or, simply, the web) is a distributed system of resources linked through hyperlinks and hosted on servers across the Internet

- Resources can be web pages, documents, multimedia, etc.
- Web pages are a critical resource as they contain hyperlinks to other resources
- Servers hosting WWW resources are called web servers
- A program that requests a resource from a web server is called a web client

WWW plumbing

When you click a hyperlink in a browser:

1. The browser (the web client in this case) parses the hyperlink to obtain
   a) the name and location of the web server hosting the associated resource
   b) the pathname of the resource on the server
2. The browser connects to the web server and sends it a message requesting the resource
3. The web server replies back with a message that includes the requested resource (if the server has it)

The technology required to do the above includes:

- Uniform Resource Locator (URL)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)

WWW plumbing

The technology required to do the above includes:

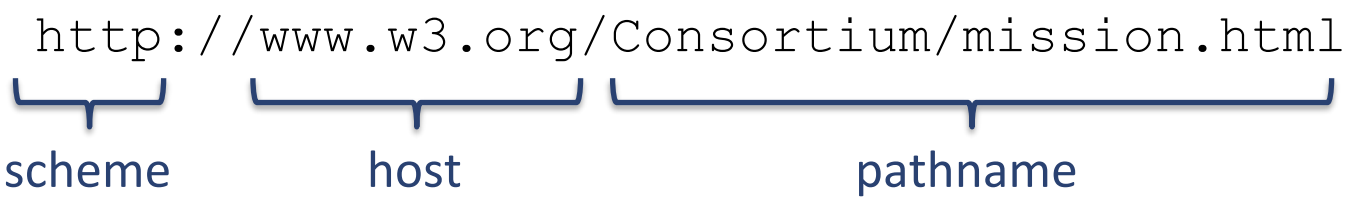
- A naming and locator scheme that uniquely identifies, and locates, resources
- A communication protocol that specifies precisely the order in which messages are sent as well as the message format
- A document formatting language for developing web pages that supports hyperlink definitions

The technology required to do the above includes:

- Uniform Resource Locator (URL)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)

Naming scheme: URL

The Uniform Resource Locator (URL) is a naming and locator scheme that uniquely identifies, and locates, resources on the web



Other examples:

- `https://webmail.cdm.depaul.edu/`
- `ftp://ftp.server.net/`
- `mailto:lperkovic@cs.depaul.edu`
- `file:///Users/lperkovic/`

Communication protocol: HTTP

The HyperText Transfer Protocol (HTTP) specifies the format of the web client's request message and the web server's reply message

Suppose the web client wants `http://www.w3.org/Consortium/mission.html`

After the client makes a network connection to internet host `www.w3.org`, it send a request message

request line
request headers

```
GET /Consortium/mission.html HTTP/1.1
Host: www.w3.org
User-Agent: Mozilla/5.0 ...
Accept: text/html,application/xhtml+xml...
Accept-Language: en-us,en;q=0.5 ...
...
```

After processing the request, the server sends back a reply that includes the requested resource (file `mission.html`)

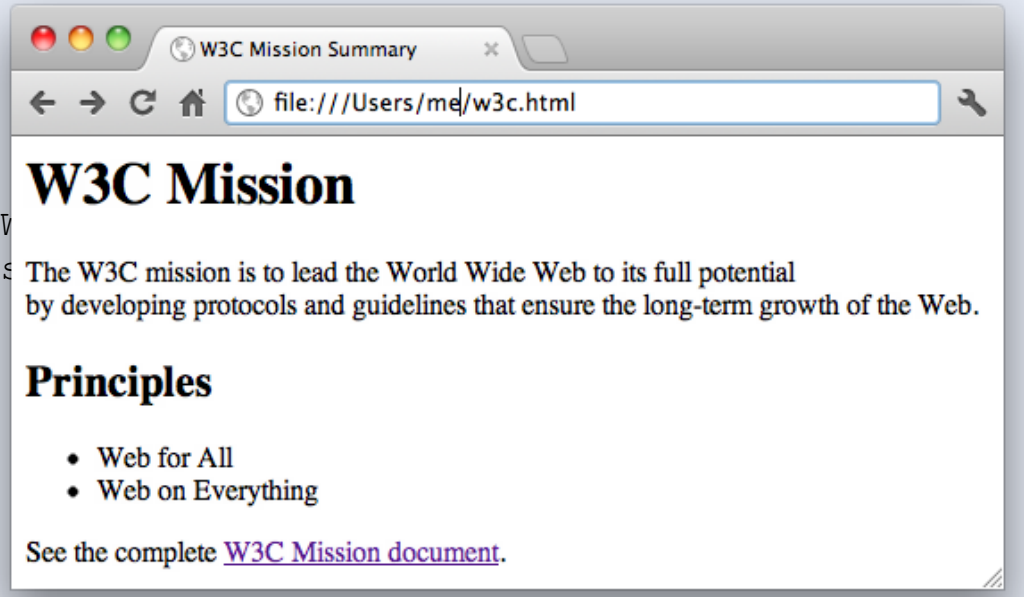
reply line
reply headers
requested resource

```
HTTP/1.1 200 OK
Date: Sat, 21 Apr 2012 16:11:26 GMT
Server: Apache/2
Last-Modified: Mon, 02 Jan 2012 17:56:24 GMT
...
Content-Type: text/html; charset=utf-8

<html>
...
</html>
```

HyperText Markup Language: HTML

An HTML file is a text file written in HTML and is referred to as the source file
The browser interprets the HTML source file and displays the web page

```
<html>
<head>
<title>W3C Mission Summary</title>
</head>
<body>
<h1>W3C Mission</h1>
<p>
The W3C mission is to lead the World W
by developing protocols and guidelines
Web.
</p>
<h2>Principles</h2>
<ul>
<li>Web for All</li>
<li>Web on Everything</li>
</ul>
See the complete <a href="http://www.w3.org/Consortium/mission.html">W3C Mission
document</a>.
</body>
</html>
```



w3c.html

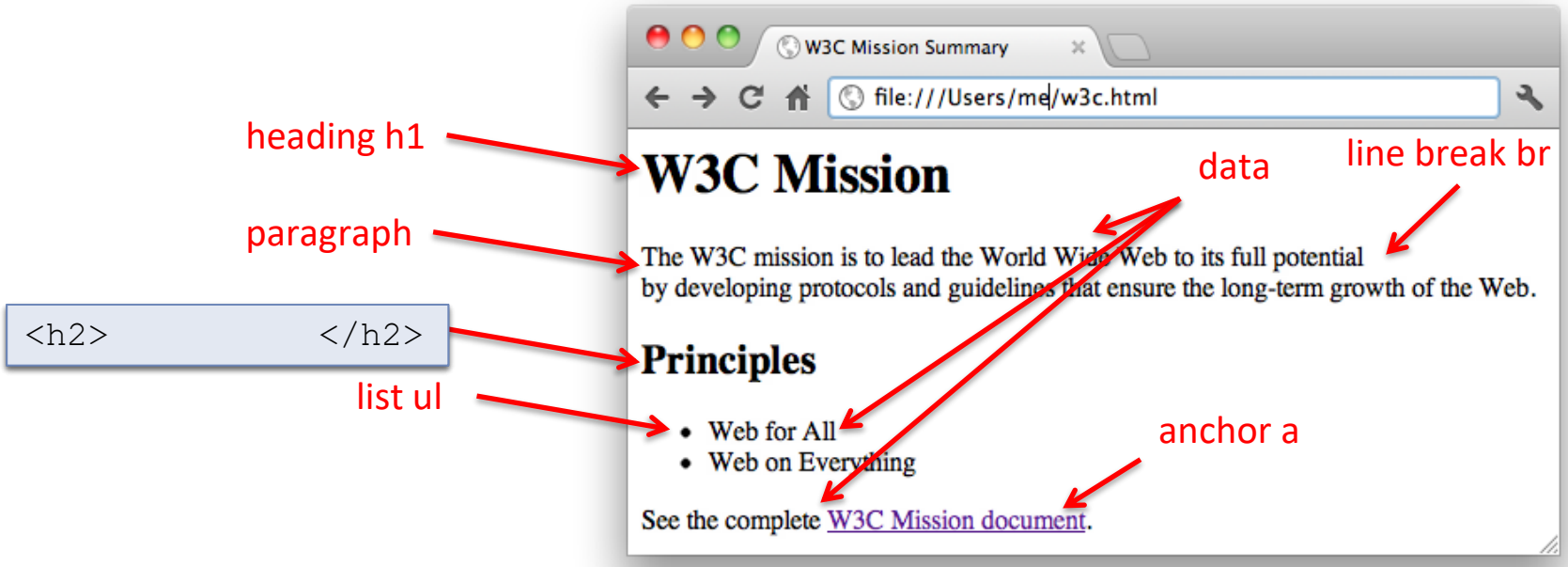
HyperText Markup Language: HTML

An HTML source file is composed of HTML elements

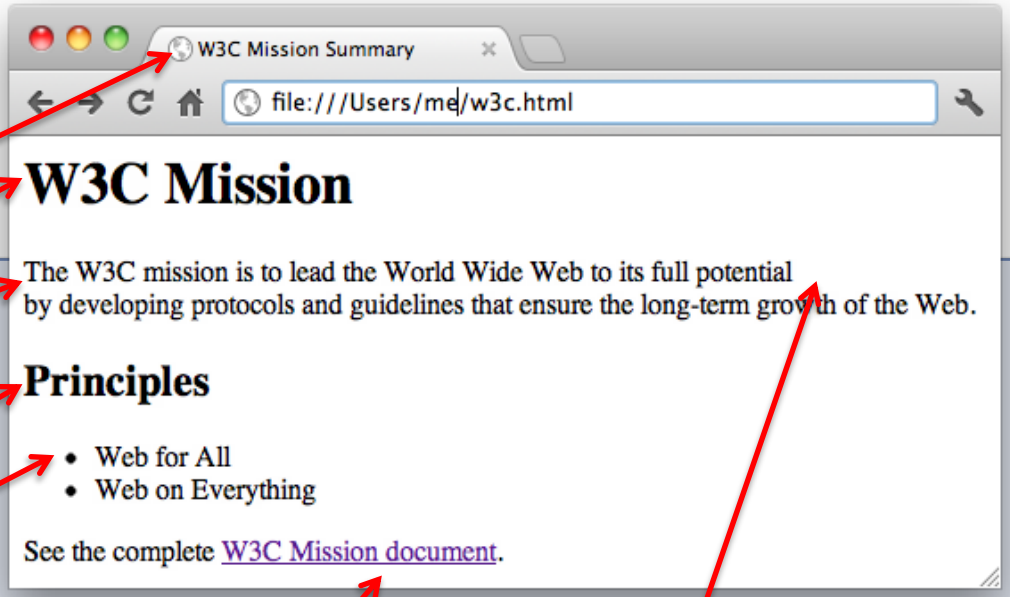
- Each element defines a component of the associated web page

In the source file, an HTML element is described using:

- A pair of tags, the start tag and the end tag
- Optional attributes within the start tag
- Other elements or data between the start and end tag



HyperText Markup Language: HTML




```
<html>
<head>
<title>W3C Mission Summary</title>
</head>
<body>
<h1>W3C Mission</h1>
<p>
The W3C mission is to lead the World Wide Web to its full potential<br>
by developing protocols and guidelines that ensure the long-term growth of the
Web.
</p>
<h2>Principles</h2>
<ul>
<li>Web for All</li>
<li>Web on Everything</li>
</ul>
See the complete <a href="http://www.w3.org/Consortium/mission.html">W3C Mission
document</a>.
</body>
</html>
```

document metadata

document content

text data

Hyperlinks

HTML anchor element (`a`) defines a hyperlink

```
<a href="http://www.w3.org/Consortium/mission.html">W3C Mission document</a>.
```

attribute | attribute value | hyperlink text

The anchor start tag must have attribute `href` whose value is a URL

The URL can be relative or absolute

```
...
<a href="/facts.html"
...
<a href="http://twitter.com/W3C"
...
```

`http://www.w3.org/Consortium/mission.html`

relative URL that corresponds to absolute URL `http://www.w3.org/Consortium/facts.html`

Python program as web client

The Python Standard Library includes several modules that allow Python to access and process web resources

Module `urllib.request` includes function `urlopen()` that takes a URL and makes a HTTP transaction to retrieve the associated resource

Python programs as web clients

```
>>> from urllib.request import urlopen
>>> response = urlopen('http://www.w3c.org/Consortium/facts.html')
>>> type(response)
<class 'http.client.HTTPResponse'>
>>> response.geturl()
'http://www.w3.org/Consortium/facts.html'
>>> for header in response.getheaders():
        print(header)

('Date', 'Mon, 23 Apr 2012 01:25:20 GMT')
('Server', 'Apache/2')
...
('Content-Type', 'text/html; charset=utf-8')
>>> html = response.read()
>>> type(html)
<class 'bytes'>
>>> html = html.decode()
>>> type(html)
<class 'str'>
>>> html
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN”
...
</div></body></html>\n'
```

`HTTPResponse` is a class defined in Standard Library module `http.client`; it encapsulates an HTTP reply.

`HTTPResponse` supports methods:

- `geturl()`
- `getheaders()`
- `read()`, etc

Because a web resource is not necessarily a text file, method `read()` returns an object of type `bytes`; `bytes` method `decode()` interprets the bytes as a string encoded in UTF-8 and returns that string

Exercise

Write method news() that takes a URL of a news web site and a list of news topics (i.e., strings) and computes the number of occurrences of each topic in the news.

```
>>> news('http://bbc.co.uk',['economy','climate','education'])
economy appears 3 times
climate appears 3 times
education appears 1 times
```

```
from urllib.request import urlopen
def news(url, topics):
    '''counts in resource with URL url the frequency
       of each topic in list topics'''

    response = urlopen(url)
    html = response.read()
    content = html.decode().lower()
    for topic in topics:
        n = content.count(topic)
        print('{} appears {} times.'.format(topic,n))
```

Parsing a web page

The Python Standard Library module `html.parser` provides a class, `HTMLParser`, for parsing HTML files.

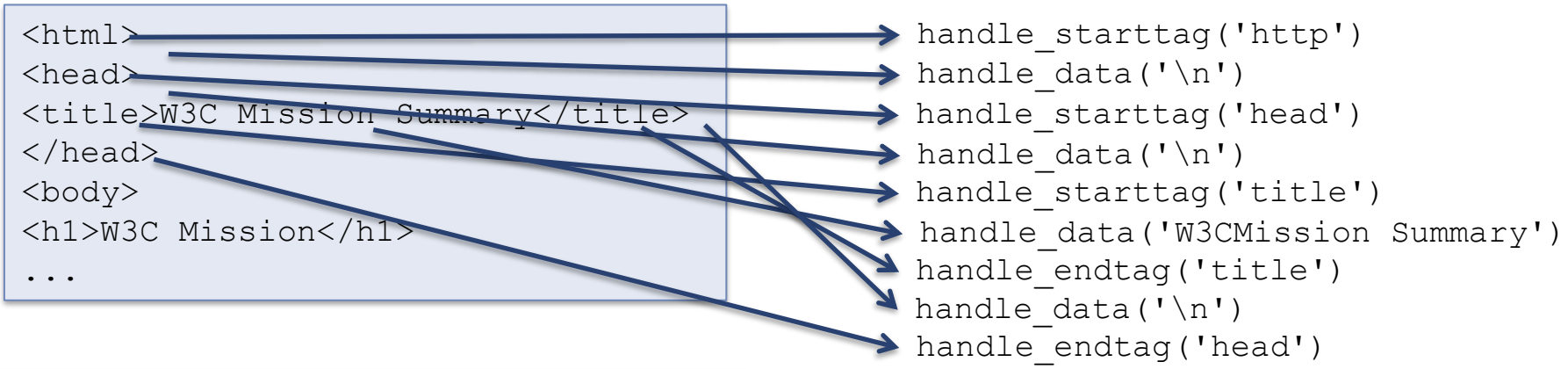
When an `HTMLParser` object is fed a string containing HTML, it processes it

```
>>> infile = open('w3c.html')
>>> content = infile.read()
>>> infile.close()
>>> from html.parser import HTMLParser
>>> parser = HTMLParser()
>>> parser.feed(content)
```

- The string content is divided into tokens that correspond to HTML start tags, end tags, text data, etc.
- The tokens are then processed in the order in which they appear in the string
- For each token, an appropriate handler is invoked by Python
- The handlers are methods of class `HTMLParser`

Token	Handler	Explanation
`<tag attrs>`	`handle_starttag(tag, attrs)`	Start tag handler
`</tag>`	`handle_endtag(tag)`	End tag handler
`data`	`handle_data(data)`	Arbitrary text data handler

Parsing a web page




```
>>> infile = open('w3c.html')
>>> content = infile.read()
>>> infile.close()
>>> from html.parser import HTMLParser
>>> parser = HTMLParser()
>>> parser.feed(content)
```

- For each token, an appropriate handler is invoked
- The handlers are methods of class `HTMLParser`
- By default, the handlers do nothing

Token	Handler	Explanation
`<tag attrs>`	`handle_starttag(tag, attrs)`	Start tag handler
`</tag>`	`handle_endtag(tag)`	End tag handler
`data`	`handle_data(data)`	Arbitrary text data handler

Parsing a web page

`HTMLParser` is really meant to be used as a "generic" superclass from which application specific parser subclasses can be developed

To illustrate, let's develop a parser that prints the URL value of the `href` attribute contained in every anchor start tag

To do this we need to create a subclass of `HTMLParser` that overrides method `handle_starttag()`

```
from html.parser import HTMLParser
class LinkParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        'print value of href attribute if any
        if tag == 'a':
            # search for href attribute and print its value
            for attr in attrs:
                if attr[0] == 'href':
                    print(attr[1])
```

ch11.html

Parsing a web page

```
<html>
<body>
<h4>Absolute HTTP link</h4>
<a href="http://www.google.com">Absolute link to Google</a>
<h4>Relative HTTP link</h4>
<a href="test.html">Relative link to test.html.</a>
<h4>mailto scheme</h4>
<a href="mailto:me@example.net">Click here to email me.</a>
</body>
</html>
```

```
>>> infile = open('links.html')
>>> content = infile.read()
>>> infile.close()
>>> linkparser = LinkParser()
>>> linkparser.feed(content)
http://www.google.com
test.html
mailto:me@example.net
```

```
from html.parser import HTMLParser
class LinkParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        'print value of href attribute if any
        if tag == 'a':
            # search for href attribute and print its value
            for attr in attrs:
                if attr[0] == 'href':
                    print(attr[1])
```

ch11.html

Exercise

Develop class `MyHTMLParser` as a subclass of `HTMLParser` that, when fed an HTML file, prints the names of the start and end tags in the order that they appear in the document, and with an indentation that is proportional to the element's depth in the document structure

```
>>> infile = open('w3c.html')
>>> content = infile.read()
>>> infile.close()
>>> myparser = MyHTMLParser()
>>> myparser.feed(content)
html start
    head start
        title start
        title end
    head end
    body start
        h1 start
        h1 end
        h2 start
...
        a end
    body end
html end
```

Exercise

```
>>> infile = open('w3c.ht
>>> content = infile.read
>>> infile.close()
>>> myparser = MyHTMLPars
>>> myparser.feed(content
html start
    head start
        title start
        title end
    head end
    body start
        h1 start
        h1 end
        h2 start
...
        a end
    body end
html end
```

```
from html.parser import HTMLParser
class MyHTMLParser(HTMLParser):
    'HTML doc parser that prints tags indented '

    def __init__(self):
        'initializes the parser and the initial
indentation'
        HTMLParser.__init__(self)
        self.indent = 0            # initial indentation
value

    def handle_starttag(self, tag, attrs):
        '''prints start tag with an indentation
proportional
           to the depth of the tag's element in the
document'''
        if tag not in {'br','p'}:
            print('{}{} start'.format(self.indent*' ',
tag))
            self.indent += 4

    def handle_endtag(self, tag):
        '''prints end tag with an indentation proportional
           to the depth of the tag's element in the
document'''
        if tag not in {'br','p'}:
```

Collecting hyperlinks (as absolute URLs)

Parser `LinkParser` prints the URL value of the `href` attribute contained in every anchor start tag

Suppose we would like to collect `http` URLS only, in their absolute version

```
>>> url = 'http://www.w3.org/Consortium/mission.html'
>>> resource = urlopen(url)
>>> content = resource.read().decode()
>>> collector = Collector()
>>> collector.feed(content)
>>> collector.getLinks()
...
http://www.w3.org/Consortium/sup
http://www.w3.org/Consortium/siteindex
http://lists.w3.org/Archives/Public/site-comments/
http://twitter.com/W3C
...
```

```
from html.parser import HTMLParser
class LinkParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        'print value of href attribute if any
        if tag == 'a':
            # search for href attribute and print its value
            for attr in attrs:
                if attr[0] == 'href':
                    print(attr[1])
```

ch11.html

Collecting hyperlinks (as absolute URLs)

Need to transform a relative URL in web page …

… to an absolute URL

The Python Standard Library module `urllib.parse` defines method `urljoin()` for this purpose

```
>>> url = 'http://www.w3.org/Consortium/mission.html'
>>> resource = urlopen(url)
>>> content = resource.read().decode()
>>> linkparser = LinkParser()
>>> linkparser.feed(content)
...
/Consortium/sup
/Consortium/siteindex
mailto:site-comments@w3.org
http://lists.w3.org/Archives/Public/site-comments/
```

```
>>> from urllib.parse import urljoin
>>> url = 'http://www.w3.org/Consortium/mission.html'
>>> relative = '/Consortium/siteindex'
>>> urljoin(url, relative)
'http://www.w3.org/Consortium/siteindex'
```

```
>>> collector = Collector()
>>> collector.feed(content)
...
http://www.w3.org/Consortium/sup
http://www.w3.org/Consortium/siteindex
http://lists.w3.org/Archives/Public/site-comments/
http://twitter.com/W3C

...
```

Collecting hyperlinks (as absolute URLs)

```
from urllib.parse import urljo
from html.parser import HTMLPa
class Collector(HTMLParser):
    'collects hyperlink URLs i

    def __init__(self, url):
        'initializes parser, t
        HTMLParser.__init__(se
        self.url = url
        self.links = []

def handle_starttag(self, tag, attrs):
        'collects hyperlink URLs in their absolute format'
        if tag == 'a':
            for attr in attrs:
                if attr[0] == 'href':
                    # construct absolute URL
                    absolute = urljoin(self.url, attr[1])
                    if absolute[:4] == 'http': # collect HTTP URLs
                        self.links.append(absolute)

    def getLinks(self):
        'returns hyperlinks URLs in their absolute format'
        return self.links
```

```
>>> url = 'http://www.w3.org/Consortium/mission.html'
>>> resource = urlopen(url)
>>> content = resource.read().decode()
>>> collector = Collector()
>>> collector.feed(content)
>>> collector.getLinks()
[...,
'http://www.w3.org/Consortium/sup',
'http://www.w3.org/Consortium/siteindex',
'http://lists.w3.org/Archives/Public/site-comments/',
'http://twitter.com/W3C',
...]
```

Regular expressions

Suppose we need to find all email addresses in a web page

- How do we recognize email addresses?
- What string pattern do emails addresses exhibit?

A email address string pattern, informally:

> An email address consists of a user ID—that is, a sequence of "allowed" characters—followed by the @ symbol followed by a hostname—that is, a dot-separated sequence of allowed characters

A regular expression is a more formal way to describe a string pattern

A regular expression is a string that consists of characters and regular expression operators

Regular expression operators

Regular expression without operators

Regular expression	Matching strings
`best`	`best`

Operator `.`

Regular expression	Matching strings
`be.t`	`best, belt, beet, bezt, be3t, be!t, be t, ...`

Operators `* + ?`

Regular expression	Matching strings
`be*t`	`bt, bet, beet, beeet, beeeet, ...`
`be+t`	`bet, beet, beeet, beeeet, ...`
`bee?t`	`bet, beet`

Regular expression operators

Operator `[]`

Regular expression	Matching strings
`be[ls]t`	`belt, best`
`be[l-o]t`	`belt, bemt, bent, beot`
`be[a-cx-z]t`	`beat, bebt, bect, bext, beyt, bezt`

Operator `^`

Regular expression	Matching strings
`be[^0-9]t`	`belt, best, be#t, ...` (but not `be4t`)
`be[^xyz]t`	`belt, be5t, ...` (but not `bext, beyt,` and `bezt`)
`be[^a-zA-Z]t`	`be!t, be5t, be t, ...` (but not `beat`)

Regular expression operators

Operator |

Regular expression	Matching strings
`hello|Hello`	`hello, Hello.`
`a+|b+`	`a, b, aa, bb, aaa, bbb, aaaa, bbbb, ...`
`ab+|ba+`	`ab,abb,abbb,...,andba,baa,baaa,...`

Regular expression operators

Operator	Interpretation
`.`	Matches any character except a new line character
`*`	Matches 0 or more repetitions of the regular expression immediately preceding it. So in regular expression `ab*`, operator `*` matches 0 or more repetitions of `b`, not `ab`
`+`	Matches 1 or more repetitions of the regular expression immediately preceding it
`?`	Matches 0 or 1 repetitions of the regular expression immediately preceding it
`[]`	Matches any character in the set of characters listed within the square brackets; a range of characters can be specified using the first and last character in the range and putting – in between
`^`	If `S` is a set or range of characters, then `[^S]` matches any character not in `S`
`|`	If `A` and `B` are regular expressions, `A|B` matches any string that is matched by `A` or `B`

Regular expression escape sequences

Regular expression operators have special meaning inside regular expressions and cannot be used to match characters `'*'`, `'.'`, or `'['`

The escape sequence \ must be used instead

- regular expression `'\*\['` matches string `'*['`

\ may also signal a regular expression special sequence

Operator	Interpretation
`\d`	Matches any decimal digit; equivalent to `[0-9]`
`\D`	Matches any nondigit character; equivalent to `[0-9]`
`\s`	Matches any whitespace character including the blank space, the tab `\r`, the new line `\r`, and the carriage return `\r`
`\S`	Matches any non-whitespace character
`\w`	Matches any alphanumeric character; this is equivalent to `[a-zA-Z0-9]`
`\W`	Matches any nonalphanumeric character; this is equivalent to `[^a-zA-Z0-9_]`

Standard Library module `re`

The Standard Library module `re` contains regular expression tools

Function `findall()` takes regular expression `pattern` and string `text` as input and returns a list of all substrings of `pattern`, from left to right, that match regular expression `pattern`

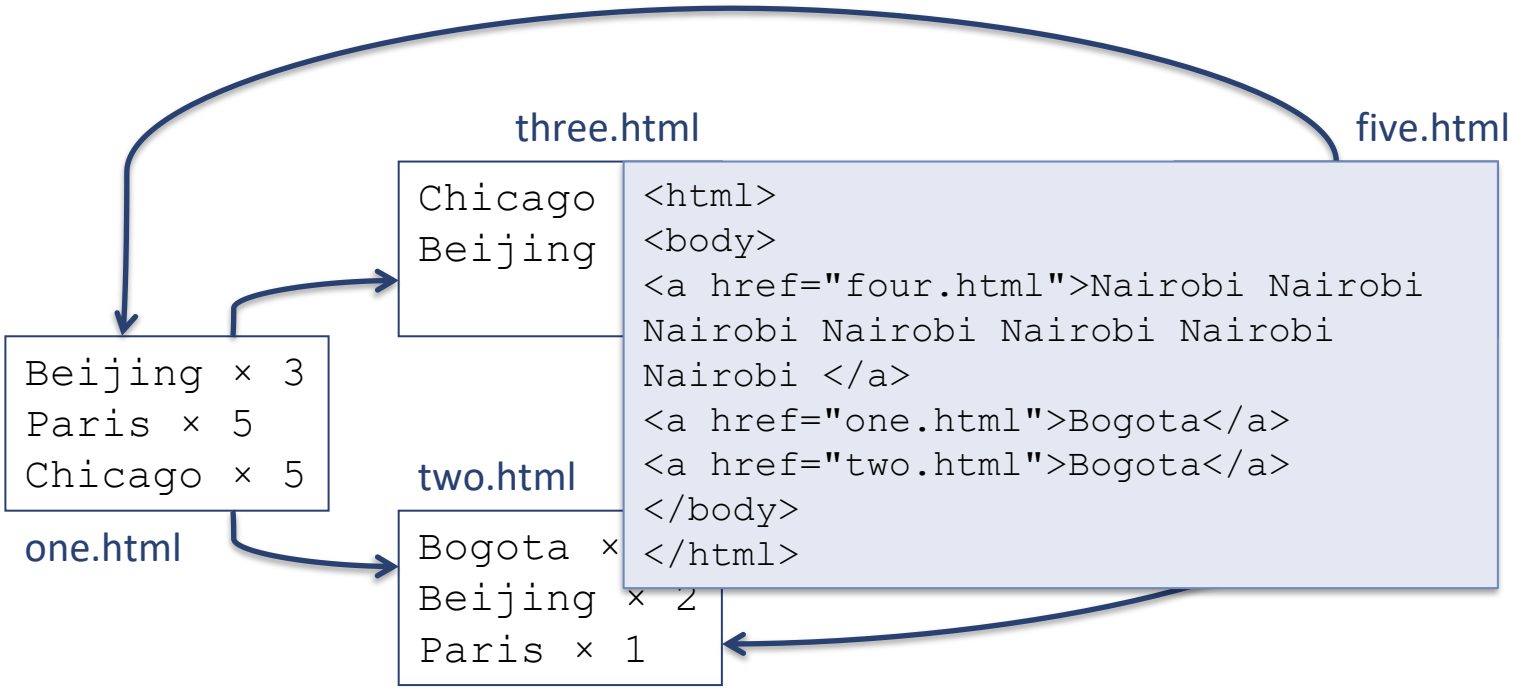
```
>>> from re import findall
>>> findall('best', 'beetbtbelt?bet, best')
['best']
>>> findall('be.t', 'beetbtbelt?bet, best')
['beet', 'belt', 'best']
>>> findall('be?t', 'beetbtbelt?bet, best')
['bt', 'bet']
>>> findall('be*t', 'beetbtbelt?bet, best')
['beet', 'bt', 'bet']
>>> findall('be+t', 'beetbtbelt?bet, best')
['beet', 'bet']
```

Case study: web crawler

A web crawler is a program that systematically visits web pages by following hyperlinks

Every time it visits a web page, a web crawler processes its content

- For example, to compute the number of occurrences of every (text data) word
- Or, to record all the hyperlinks in the web page



Case study: web crawler

A very simple crawler can be described using recursion

When the crawler is called on a URL

1. Analyze associated web page
2. Recursively call crawler on every hyperlink

```
def crawl1(url):
    'recursive web crawler that calls analyze() on every web page'

    # analyze() returns a list of hyperlink URLs in web page url
    links = analyze(url)

    # recursively continue crawl from every link in links
    for link in links:
        try:  # try block because link may not be valid HTML file
            crawl1(link)
        except:             # if an exception is thrown,
            pass            # ignore and move on.
```

Case study: web crawler

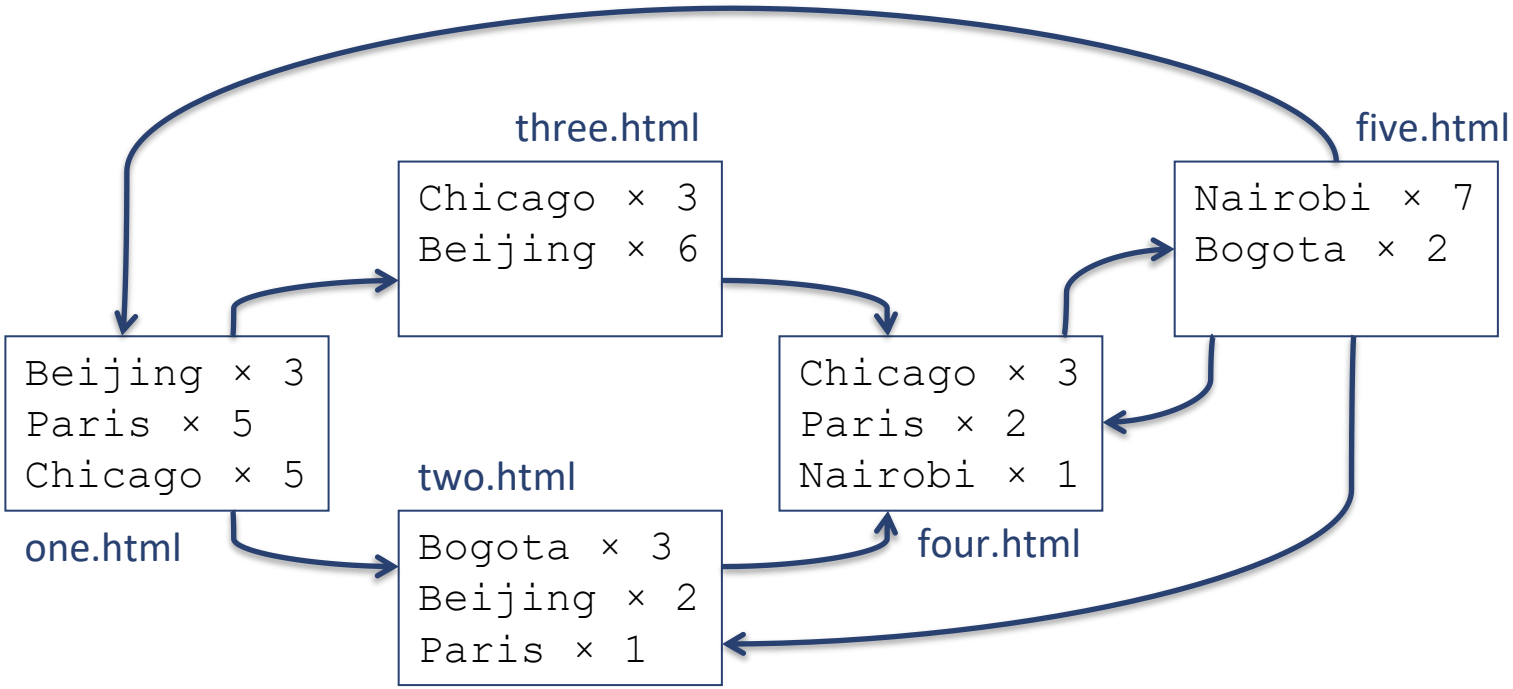
```
from urllib.request import urlopen
def analyze(url):
    'returns list of http links in url, in absolute format'

    print('\n\nVisiting', url)           # for testing

    # obtain links in the web page
    content = urlopen(url).read().decode()
    collector = Collector(url)
    collector.feed(content)
    urls = collector.getLinks()          # get list of links

    return urls
```

```
def crawl1(url):
    'recursive web crawler that calls analyze() on every web page'

    # analyze() returns a list of hyperlink URLs in web page url
    links = analyze(url)

    # recursively continue crawl from every link in links
    for link in links:
        try:  # try block because link may not be valid HTML file
            crawl1(link)
        except:             # if an exception is thrown,
            pass            # ignore and move on.
```

Case study: web crawler

Problem: the crawler visits some pages infinitely often and some not at all

The crawler should ignore the links to pages it has already visited; to do this, we need to keep track of visited pages



Case study: web crawler

```
>>> crawl22('http://reed.cs.depaul.edu/lperkovic/one.html')
Visiting http://reed.cs.depaul.edu/lperkovic/one.html
Visiting http://reed.cs.depaul.edu/lperkovic/two.html
Visiting http://reed.cs.depaul.edu/lperkovic/four.html
Visiting http://reed.cs.depaul.edu/lperkovic/five.html
Visiting http://reed.cs.depaul.edu/lperkovic/three.html
```

```
visited = set() # i

def crawl2(url):
    '''a recursive web crawler that calls analyze()
       on every visited web page'''

    # add url to set of visited pages
    global visited     # warns the programmer
    visited.add(url)

    # analyze() returns a list of hyperlink URLs in web page url
    links = analyze(url)

    # recursively continue crawl from every link in links
    for link in links:
        # follow link only if not visited
        if link not in visited:
            try:
                crawl2(link)
            except:
                pass
```

Case study: web crawler

```
from urllib.request import urlopen
def analyze(url):

    print('\n\nVisiting', url)           # for testing

    # obtain links in the web page
    content = urlopen(url).read().decode()
    collector = Collector(url)
    collector.feed(content)
    urls = collector.getLinks()          # get list of links

    # compute word frequencies
    content = collector.getData()
    freq = frequency(content)

    # print the frequency of every text data word in web page
    print('\n{:45} {:10} {:5}'.format('URL', 'word', 'count'))
    for word in freq:
        print('{:45} {:10} {:5}'.format(url, word, freq[word]))

    # print the http links found in web page
    print('\n{:45} {:10}'.format('URL', 'link'))
    for link in urls:
        print('{:45} {:10}'.format(url, link))

    return urls
```

Case study: web crawler

```
>>> crawl2('http://reed.cs.depaul.edu/lperkovic/one.html')

Visiting http://reed.cs.depaul.edu/lperkovic/one.html

URL                                          word       count
http://reed.cs.depaul.edu/lperkovic/one.html  Paris          5
http://reed.cs.depaul.edu/lperkovic/one.html  Beijing        3
http://reed.cs.depaul.edu/lperkovic/one.html  Chicago        5

URL                                          link
http://reed.cs.depaul.edu/lperkovic/one.html  http://reed.cs.depaul.edu/lperkovic/two.html
http://reed.cs.depaul.edu/lperkovic/one.html  http://reed.cs.depaul.edu/lperkovic/three.html


Visiting http://reed.cs.depaul.edu/lperkovic/two.html

URL                                          word       count
http://reed.cs.depaul.edu/lperkovic/two.html  Bogota         3
http://reed.cs.depaul.edu/lperkovic/two.html  Paris          1
http://reed.cs.depaul.edu/lperkovic/two.html  Beijing        2

URL                                          link
http://reed.cs.depaul.edu/lperkovic/two.html  http://reed.cs.depaul.edu/lperkovic/four.html


Visiting http://reed.cs.depaul.edu/lperkovic/four.html

URL                                          word       count
http://reed.cs.depaul.edu/lperkovic/four.html Paris          2
...
```