CSC 242 Section 504 Winter 2016
Python Web Tools (Chapter 11)

- World Wide Web
- Python Modules for WWW
- Web Crawling
- String Pattern Matching
The World Wide Web (WWW)

The Internet connects computers around the world

• A web server is a computer which hosts web sites
  • “Server” may also refer to the program that resides on the server computer
• A web client is a program that communicates with a web server, such as Internet Explorer or other web browsers
  • Example: Web browsers such as Firefox, Internet Explorer, etc.
  • Other programs as well
• We will write web client applications
HyperText Markup Language: HTML

Many web pages are written in a language called HTML (HyperText Markup Language).

An HTML file is composed of HTML elements:
- Each element defines a component of the associated web page.

An HTML element consists of:
- A pair of tags, usually a start tag and an end tag.
- Tag name is within `< ... >`.
- Optional attributes within the start tag.
- Other elements or data between the start and end tag.
HTML Example

<title>Simple web page</title>
</head>
<body>
<h3>Here is a very simple web page.</h3>
<ul>
<li>It has an unordered list</li>
<li>This is <b>boldface</b>, this is <em>italics</em></li>
</ul>
<ol>
<li>Here is an embedded ordered List</li>
<p>A separate paragraph</p>
<li>Another item</li>
</ol>
</body>
</html>
Start and end tags

All tags start with < and end with >; for example <ul>

End tags are of the form </ul>

For some elements, the end tag is optional, or not allowed

<p>
Here is a paragraph; the end tag is optional
With <br>, no end tag is allowed
Attributes

Tags also may contain **attributes**, which specify additional information

```html
<table border=1>
<tr><th>Score</th><th>Grade</th></tr>
<tr><td>90-100</td><td align=center>A</td></tr>
<tr><td colspan=2>Et cetera</td></tr>
</table>
```
Hyperlinks

One way to create a hyperlink is by using the anchor tag

```html
<html>
<body>
Here is an example of a page with a hyperlink: <a href="http://condor.depaul.edu/slytinen/242w16">242 condor.depaul.edu homepage</a>
</body>
</html>
```
# The Anchor tag

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;a&gt;</code></td>
<td>Defines a link to another URL. When the user clicks on the text that’s displayed by the tag, the browser requests the page that is identified by the Href attribute of the tag.</td>
</tr>
</tbody>
</table>

## One attribute of the Anchor tag

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>href</code></td>
<td>Specifies the URL for the link.</td>
</tr>
</tbody>
</table>
Exercise

Write an HTML file which is displayed as follows by a browser
Hyperlinks

The value of an anchor tag’s `href` attribute is a **Universal Resource Locator (URL)**

The URL may be **absolute** or **relative**

```html
<a href="http://condor.depaul.edu/slytinen/242w16">242 homepage</a><br>
<a href="hyperlink.html">hyperlink example</a><br>
<a href="/">condor root</a><br>
<a href="/simple.html">another way to access simple.html</a><br>
<a href="../exercises/web.py">Web exercises</a>
```
Example use of an img tag

```html
<html>
<body>
<table>
<tr><td align=center><h3>CSC 242 Text</h3></td></tr>
<tr><td width=100><img src="PerkovicPython.jpg"></a></td></tr>
</table>
</body>
</html>
```
Exercise

Write an HTML file which is displayed as follows by a browser
Naming scheme: URL

Web pages are accessed through their Uniform Resource Locator (URL). This naming scheme that uniquely identifies, and locates, resources on the web

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

scheme host pathname

Other examples:

• https://webmail.cdm.depaul.edu/
• ftp://ftp.server.net/
• mailto:lperkovic@cs.depaul.edu
HTTP and Python

• The `urllib.request` module contains a method called `urlopen`

  Example:
  `urlopen('http://condor.depaul.edu/slytinen/242w16/index.html')`

• The `urlopen` function generates an HTTP request that is sent to the specified Web server

  In the above example, the server’s name is `condor.depaul.edu`

• The returned object has a `read` method
The read() method returns an object whose type is binary (not str).

This is because not all resources on the Web are textual – for example, multimedia files.

If a resource is text (for example a Web page), then the decode() method must be called to convert from binary to str.

```python
>>> from urllib.request import urlopen
>>> resource = urlopen('http://condor.depaul.edu/slytinen/242w16')
>>> contents = resource.read().decode()
```
Example

The function below `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.

```python
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))
```

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

Write a function called `images` which is passed a web page’s URL, and prints the URLs all images that are on the page. Assume a URL appears immediately after `src=` and ends before `>` or `'` `'`. Strip the “…” from the beginning and end of the URL if they are there.

```python
>>> images('http://www.cdm.depaul.edu/)
/PublishingImages/logoCDM.png" alt="DePaul University" id="mainLogo" /
/PublishingImages/logocdmshield.png" alt="DePaul University" id="altLogo" /
/PublishingImages/printLogo.gif" alt="DePaul University | College of Computing and Digital Media" /
../publishingimages/hero-home-sca.jpg" alt="College of Computing and Digital Media's School of Cinematic Arts" /
../publishingimages/hero-home-soc.jpg" alt="College of Computing and Digital Media's School of Computing" /
../publishingimages/hero-home-sod.jpg" alt="College of Computing and Digital Media's School of Design" /
/PublishingImages/facebook.svg" border="0" /
/PublishingImages/twitter.svg" border="0" /
/PublishingImages/linkedin.svg" border="0" /
```
def images(url):
    contents = readurl(url)
    try:
        while True:
            start = contents.index('<img src=')
            contents = contents[start+9:]
            end = contents.index('>')
            blank = contents.find(' ')
            if blank == -1:
                blank = len(contents)
            imgurl = contents[:min(end,blank)].strip('"
')
            print(imgurl)
            contents = contents[end:]
            # index can raise a ValueError
    except ValueError:
        return
Exercise

Write a function called `hyperlinks`. The function is passed a URL, and should return a list of URLs on that page that appear as hyperlinks.

```python
>>> hyperlinks('http://condor.depaul.edu/slytinen/242w16/web')
[ ‘example1.html’, ‘hyperlink.html’, … ]
```

Note that these are not complete URLs
Exercise

def hyperlinks(content):
    links = []
    while True:
        try:
            urlstart = content.index('href=') + 5
            urlend = content.index('>', urlstart)
            link = content[urlstart:urlend].strip('""
            links.append(content[urlstart:urlend])
        except ValueError:
            return links
        content = content[urlend:]
Parsing a web page

A parser is a program that analyzes the structure of a piece of text.

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

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

- The string is broken 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
HTMLParser methods

```python
>>> from urllib.request import urlopen
>>> from html.parser import HTMLParser
>>> response = urlopen('http://condor.depaul.edu/slytinen/242w16')
>>> content = response.read().decode()
>>> parser = HTMLParser()
>>> parser.feed(content)
```

<table>
<thead>
<tr>
<th>Token</th>
<th>HTMLParser method</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;tag attrs&gt;</code></td>
<td>handle_starttag(tag, attrs)</td>
<td>Start tag handler</td>
</tr>
<tr>
<td><code>&lt;/tag&gt;</code></td>
<td>handle_endtag(tag)</td>
<td>End tag handler</td>
</tr>
<tr>
<td><code>data</code></td>
<td>handle_data(data)</td>
<td>Arbitrary text data handler</td>
</tr>
</tbody>
</table>

Everything on a Web page is a start tag, an end tag, or data.

Using HTMLParser, it will be handled by one of the 3 methods above.
Parsing a web page

```python
>>> from html.parser import HTMLParser
>>> response = urllib.urlopen('http://www.w3c.org')
>>> content = response.read().decode()
>>> 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

<table>
<thead>
<tr>
<th>Token</th>
<th>Handler</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tag attrs&gt;</td>
<td>handle_starttag(tag, attrs)</td>
<td>Start tag handler</td>
</tr>
<tr>
<td>&lt;/tag&gt;</td>
<td>handle_endtag(tag)</td>
<td>End tag handler</td>
</tr>
<tr>
<td>data</td>
<td>handle_data(data)</td>
<td>Arbitrary text data</td>
</tr>
</tbody>
</table>
HTMLParser is really meant to be used as a “generic” superclass from which application specific parser subclasses can be developed.

Here is an example subclass:

class HTMLDisplay(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print('Start tag: {} attributes: {}'.format(tag, attrs))

    def handle_endtag(self, tag):
        print('End tag: {}'.format(tag))

    def handle_data(self, data):
        print('Data: {}'.format(data))
CSC 242 504 Web page
A web page
Hello world!
CSC 242-504 homepage
<html>
<title>CSC 242 504 Web page</title>
<body>
<center>
<h3>CSC 242 504</h3>
A web page
</center>
<p>Hello world!</p>
<p>
<a href="http://condor.depaul.edu/slytinen/242w16/"
>CSC 242-504 homepage</a>
</p>
</body>
</html>
A parser that prints hyperlink URLs

We need a parser that finds the URL attribute of every anchor start tag.

As with the previous example, we create a subclass of HTMLParser, and override the handle_starttag method.

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:
                # each attr is a tuple
                if attr[0] == 'href':
                    print(attr[1])
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:
                # each attr is a tuple
                if attr[0] == 'href':
                    print(attr[1])

Exercise: Write a function print_links which is passed a URL as a parameter and uses LinkParser to print the hyperlinks on that URL
Crawling the Web

Exercise: Complete the Crawler class in the web.py file
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.

```python
>>> url = ...
>>> resource = urlopen(url)
>>> content = resource.read().decode()
>>> collector = Collector()
>>> collector.feed(content)
>>> collector.getLinks()
...
http://www.google.com
```

Need to convert to absolute URL:

```html
<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>
<img src="image.jpg" >
</body>
</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

```python
>>> url = 'http://www.w3.org/Consortium/mission.html'
>>> resource = urlopen(url)
>>> content = resource.read().decode()
>>> linkparser = LinkParser()
>>> linkparser.feed(content)

... http://www.w3.org/Consortium/sup http://www.w3.org/Consortium/siteindex http://mailto:site-comments@w3.org http://lists.w3.org/Archives/Public/site-comments/ http://twitter.com/W3C ...
```

```python
>>> 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'
```

```python
>>> content = resource.read().decode()
>>> 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 ...
```
from urllib.parse import urljoin
from html.parser import HTMLParser

class Collector(HTMLParser):
    'collects hyperlink URLs into a list'

    def __init__(self, url):
        'initializes parser, the url, and a list'
        HTMLParser.__init__(self)
        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':
                    absolute = urljoin(self.url, attr[1])
                    if absolute[:4] == 'http':
                        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

Example task: find all email addresses in a web page

Some are marked by `<a href="mailto: ... " >`

- Although even then, there may be additional characters at the end

```python
>>> print_emails('http://www.cdm.depaul.edu/about/Pages/People/Administration.aspx
... wwwfeedback@cdm.depaul.edu?subject=[www feedback] page visited: /about/pages/peo
  • Can we recognize email addresses more accurately? - yes
  • What string pattern do emails addresses exhibit?
  • Can be characterized using Python regular expression module
```
Regular expressions

- Can we recognize email addresses more accurately? - yes
- What string pattern do email addresses exhibit?
- Can be characterized using Python regular expression module

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

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>be.t</td>
<td>best, belt, beet, bezt, be3t, belt, be t, ...</td>
</tr>
</tbody>
</table>

**Operators** * + ?

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>be*t</td>
<td>bt, bet, beet, beeet, beeeet, ...</td>
</tr>
<tr>
<td>be+t</td>
<td>bet, beet, beeet, beeeet, ...</td>
</tr>
<tr>
<td>bee?t</td>
<td>bet, beet</td>
</tr>
</tbody>
</table>
# Regular expression operators

## Operator []

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>be[ls]t</td>
<td>belt, best</td>
</tr>
<tr>
<td>be[l-o]t</td>
<td>belt, bemt, bent, beot</td>
</tr>
<tr>
<td>be[a-cx-z]t</td>
<td>beat, bebt, bect, bext, beyt, bezt</td>
</tr>
</tbody>
</table>

## Operator ^

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>be[^0-9]t</td>
<td>belt, best, be#t, ... (but not be4t)</td>
</tr>
<tr>
<td>be[^xyz]t</td>
<td>belt, be5t, ... (but not bext, beyt, and bezt)</td>
</tr>
<tr>
<td>be[^a-zA-Z]t</td>
<td>belt, be5t, be t, ... (but not beat)</td>
</tr>
</tbody>
</table>
## Regular expression operators

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Matching strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>Hello</td>
</tr>
<tr>
<td>a+</td>
<td>b+</td>
</tr>
<tr>
<td>ab+</td>
<td>ba+</td>
</tr>
</tbody>
</table>
### Regular expression operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Matches any character except a new line character</td>
</tr>
<tr>
<td>*</td>
<td>Matches 0 or more repetitions of the regular expression immediately preceding it. So in regular expression <code>ab*</code>, operator * matches 0 or more repetitions of <code>b</code>, not <code>ab</code></td>
</tr>
<tr>
<td>+</td>
<td>Matches 1 or more repetitions of the regular expression immediately preceding it</td>
</tr>
<tr>
<td>?</td>
<td>Matches 0 or 1 repetitions of the regular expression immediately preceding it</td>
</tr>
<tr>
<td>[]</td>
<td>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</td>
</tr>
<tr>
<td>^</td>
<td>If S is a set or range of characters, then <code>[^S]</code> matches any character not in S</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Regular expression escape sequence

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>Matches any decimal digit; equivalent to [0-9]</td>
</tr>
<tr>
<td>\D</td>
<td>Matches any nondigit character; equivalent to [0-9]</td>
</tr>
<tr>
<td>\s</td>
<td>Matches any whitespace character including the blank space, the tab \r, the newline \r, and the carriage return \r</td>
</tr>
<tr>
<td>\S</td>
<td>Matches any non-whitespace character</td>
</tr>
<tr>
<td>\w</td>
<td>Matches any alphanumeric character; this is equivalent to [a-zA-Z0-9]</td>
</tr>
<tr>
<td>\W</td>
<td>Matches any nonalphanumeric character; this is equivalent to [^a-zA-Z0-9_]</td>
</tr>
</tbody>
</table>
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

```python
code
>>> from re import findall
code
>>> findall('best', 'beetbttbet?bet, best')
['best']
code
>>> findall('be.t', 'beetbttbet?bet, best')
['beet', 'belt', 'best']
code
>>> findall('be?t', 'beetbttbet?bet, best')
['bt', 'bet']
code
>>> findall('be+t', 'beetbttbet?bet, best')
['beet', 'bet']
```
Reg exp for e-mail addresses

• E-mail: ... @ ... .com or ... @ ... .edu
• No blanks

```python
>>> from re import findall
>>> findall(pattern, 'lytinen@cs.depaul.edu, cdm@cdm.depaul.edu')
['lytinen@cs.depaul.edu', 'cdm@cdm.depaul.edu']
```

• What is the pattern?
Python Iterators

• Loops and other constructs implicitly use Python iterators

• Example: simple for loop

```python
for item in items:
    print(item)
```

• Under the hood, here’s what’s going on:

```python
i = iter(items)
try:
    while (True):
        item = next(i)
        print(item)
except StopIteration:
    pass
```
Constructing your own iterator

• User-defined classes can behave as iterators

• Python “magic methods” __iter__, __next__

• Example:

class ItEx:
    def __init__(self, max):
        self.max = max
        self.count = -1

    def __iter__(self):
        return self

    def __next__(self):
        self.count += 1
        if self.count == self.max:
            raise StopIteration()
        return self.count
Constructing your own iterator

• Example: Create a class called **Bond**. The class should behave as follows:

  • **Constructor**: passed self and 3 parameters:
    • Initial balance
    • Duration (e.g., 30 years)
    • Interest rate (e.g., .03)

  • Behavior:

    • In a loop, amount is balance after the next year
    • `b = Bond(100, 30, .03)`
    • `for amt in b:`
      • `print(amt)`
__getitem__

- Magic method that overloads [ ]
- Behavior:

```python
b = Bond(100, 30, .03)
b[10]
for i in range(30):
    print(b[i])
```