Parsing HTML

the `HTMLParser` module

Tallying the Tags in an HTML page

overriding methods in HTMLParser

Getting the Links a Page refers to

filtering attributes of tags

Selecting certain Files

downloading only files of certain type

Getting Text formatted in Bold

the parser class exports a switch

MCS 275 Lecture 38
Programming Tools and File Management
Jan Verschelde, 18 April 2008

the `HTMLParser` module

to parse html code

In the standard Python distribution:

```
>>> import HTMLParser
>>> help(HTMLParser)
```

the class HTMLParser

- allows to override handlers of tags
- provides a `feed` method

$\rightarrow$ the `feed` method handles buffering

Using HTMLParser

```
from HTMLParser import HTMLParser
from urllib import urlopen

class OurHTMLParser(HTMLParser):

    def __init__(self):

    def handle_starttag(self, tag, attrs):

    def handle_endtag(self, tag):

def main():
    f = urlopen(page)
    p = OurHTMLParser()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
```

Using HTMLParser

```
from HTMLParser import HTMLParser
from urllib import urlopen

class OurHTMLParser(HTMLParser):

    def __init__(self):

    def handle_starttag(self, tag, attrs):

    def handle_endtag(self, tag):

def main():
    f = urlopen(page)
    p = OurHTMLParser()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
```

Using HTMLParser

```
from HTMLParser import HTMLParser
from urllib import urlopen

class OurHTMLParser(HTMLParser):

    def __init__(self):

    def handle_starttag(self, tag, attrs):

    def handle_endtag(self, tag):

def main():
    f = urlopen(page)
    p = OurHTMLParser()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
```

Using HTMLParser

```
from HTMLParser import HTMLParser
from urllib import urlopen

class OurHTMLParser(HTMLParser):

    def __init__(self):

    def handle_starttag(self, tag, attrs):

    def handle_endtag(self, tag):

def main():
    f = urlopen(page)
    p = OurHTMLParser()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
```

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Tallying the Tags in an HTML page

using the class HTMLParser

Gather basic statistics about a page:

1. what types of tags are used,
2. count number of occurrences for each tag.

At end of each tag the tally is updated.

Data structure for the tally: dictionary.

- keys: string with type of tag
- values: natural number counts #occurrences

The tally is an object data attribute.

Parsing HTML

the Class TagTally

```
from HTMLParser import HTMLParser
from urllib import urlopen

class TagTally(HTMLParser):
    """
    Makes a tally of ending tags.
    """
    def __init__(self):
        """
        Initializes the dictionary of tags.
        """
    def handle_endtag(self, tag):
        """
        Maintains a tally of the tags.
        """
    def ShowTally(self):
        """
        Prints the tally to screen.
        """
```

the Class TagTally

```
from HTMLParser import HTMLParser
from urllib import urlopen

class TagTally(HTMLParser):
    """
    Makes a tally of ending tags.
    """
    def __init__(self):
        """
        Initializes the dictionary of tags.
        """
    def handle_endtag(self, tag):
        """
        Maintains a tally of the tags.
        """
    def ShowTally(self):
        """
        Prints the tally to screen.
        """
```

the Class TagTally

```
from HTMLParser import HTMLParser
from urllib import urlopen

class TagTally(HTMLParser):
    """
    Makes a tally of ending tags.
    """
    def __init__(self):
        """
        Initializes the dictionary of tags.
        """
    def handle_endtag(self, tag):
        """
        Maintains a tally of the tags.
        """
    def ShowTally(self):
        """
        Prints the tally to screen.
        """
```

the Class TagTally

```
from HTMLParser import HTMLParser
from urllib import urlopen

class TagTally(HTMLParser):
    """
    Makes a tally of ending tags.
    """
    def __init__(self):
        """
        Initializes the dictionary of tags.
        """
    def handle_endtag(self, tag):
        """
        Maintains a tally of the tags.
        """
    def ShowTally(self):
        """
        Prints the tally to screen.
        """
```

the Class TagTally

```
from HTMLParser import HTMLParser
from urllib import urlopen

class TagTally(HTMLParser):
    """
    Makes a tally of ending tags.
    """
    def __init__(self):
        """
        Initializes the dictionary of tags.
        """
    def handle_endtag(self, tag):
        """
        Maintains a tally of the tags.
        """
    def ShowTally(self):
        """
        Prints the tally to screen.
        """
```

the Function `main()`

of script `tallytags.py`

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = TagTally()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'the tally of tags :'
    p.ShowTally()
```

the Function `main()`

of script `tallytags.py`

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = TagTally()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'the tally of tags :'
    p.ShowTally()
```

the Function `main()`

of script `tallytags.py`

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = TagTally()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'the tally of tags :'
    p.ShowTally()
```

Constructor and ShowTally

of the class TagTally

If overriding __init__, we must initialize parent class.

```
def __init__(self):
    """
    Initializes the dictionary of tags.
    """
    HTMLParser.__init__(self)
    self.TagTally = {}

def ShowTally(self):
    """
    Prints the tally to screen.
    """
    for each in self.TagTally:
        print each, ':', self.TagTally[each]
```

Constructor and ShowTally

of the class TagTally

If overriding __init__, we must initialize parent class.

```
def __init__(self):
    """
    Initializes the dictionary of tags.
    """
    HTMLParser.__init__(self)
    self.TagTally = {}

def ShowTally(self):
    """
    Prints the tally to screen.
    """
    for each in self.TagTally:
        print each, ':', self.TagTally[each]
```

Constructor and ShowTally

of the class TagTally

If overriding __init__, we must initialize parent class.

```
def __init__(self):
    """
    Initializes the dictionary of tags.
    """
    HTMLParser.__init__(self)
    self.TagTally = {}

def ShowTally(self):
    """
    Prints the tally to screen.
    """
    for each in self.TagTally:
        print each, ':', self.TagTally[each]
```

Update of the Tally

updating a dictionary

First check if there is already a tag...

```
def handle_endtag(self, tag):
    """
    Maintains a tally of the tags.
    """
    if self.TagTally.has_key(tag):
        self.TagTally[tag] \
            = self.TagTally[tag] + 1
    else:
        self.TagTally.update({tag:1})
```

If no tag present, update the dictionary.

Update of the Tally

updating a dictionary

First check if there is already a tag...

```
def handle_endtag(self, tag):
    """
    Maintains a tally of the tags.
    """
    if self.TagTally.has_key(tag):
        self.TagTally[tag] \
            = self.TagTally[tag] + 1
    else:
        self.TagTally.update({tag:1})
```

If no tag present, update the dictionary.

Update of the Tally

updating a dictionary

First check if there is already a tag...

```
def handle_endtag(self, tag):
    """
    Maintains a tally of the tags.
    """
    if self.TagTally.has_key(tag):
        self.TagTally[tag] \
            = self.TagTally[tag] + 1
    else:
        self.TagTally.update({tag:1})
```

If no tag present, update the dictionary.

Update of the Tally

updating a dictionary

First check if there is already a tag...

```
def handle_endtag(self, tag):
    """
    Maintains a tally of the tags.
    """
    if self.TagTally.has_key(tag):
        self.TagTally[tag] \
            = self.TagTally[tag] + 1
    else:
        self.TagTally.update({tag:1})
```

If no tag present, update the dictionary.

Get Links a Page refers to

Recall our code for a web crawler:

- double quoted strings starting with http

could be misleading at times, cumbersome code.

A more proper way to get the hyperlinks:

1. look for tags of type `'a'`
2. name of attribute is `href`
3. get hyperlink corresponding to `href`

Get Links a Page refers to

Recall our code for a web crawler:

- double quoted strings starting with http

could be misleading at times, cumbersome code.

A more proper way to get the hyperlinks:

1. look for tags of type `'a'`
2. name of attribute is `href`
3. get hyperlink corresponding to `href`

Get Links a Page refers to

Recall our code for a web crawler:

- double quoted strings starting with http

could be misleading at times, cumbersome code.

A more proper way to get the hyperlinks:

1. look for tags of type `'a'`
2. name of attribute is `href`
3. get hyperlink corresponding to `href`

Get Links a Page refers to

Recall our code for a web crawler:

- double quoted strings starting with http

could be misleading at times, cumbersome code.

A more proper way to get the hyperlinks:

1. look for tags of type `'a'`
2. name of attribute is `href`
3. get hyperlink corresponding to `href`

Parsing HTML

the Class HTMLrefs

```
from HTMLParser import HTMLParser
from urllib import urlopen

class HTMLrefs(HTMLParser):
    """
    Makes a list of all html links.
    """
    def __init__(self):
        """
        Initializes the list of links.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'a' and
        stores links for href attributes.
        """
    def ShowRefs(self):
        """
        Prints the HTML refs to screen.
        """
```

the Class HTMLrefs

```
from HTMLParser import HTMLParser
from urllib import urlopen

class HTMLrefs(HTMLParser):
    """
    Makes a list of all html links.
    """
    def __init__(self):
        """
        Initializes the list of links.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'a' and
        stores links for href attributes.
        """
    def ShowRefs(self):
        """
        Prints the HTML refs to screen.
        """
```

the Class HTMLrefs

```
from HTMLParser import HTMLParser
from urllib import urlopen

class HTMLrefs(HTMLParser):
    """
    Makes a list of all html links.
    """
    def __init__(self):
        """
        Initializes the list of links.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'a' and
        stores links for href attributes.
        """
    def ShowRefs(self):
        """
        Prints the HTML refs to screen.
        """
```

the Class HTMLrefs

```
from HTMLParser import HTMLParser
from urllib import urlopen

class HTMLrefs(HTMLParser):
    """
    Makes a list of all html links.
    """
    def __init__(self):
        """
        Initializes the list of links.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'a' and
        stores links for href attributes.
        """
    def ShowRefs(self):
        """
        Prints the HTML refs to screen.
        """
```

the Function main()

of the script htmlrefs.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = HTMLrefs()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'all html links :'
    p.ShowRefs()
```

the Function main( )

of the script htmlrefs.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = HTMLrefs()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'all html links :'
    p.ShowRefs()
```

the Function main( )

of the script htmlrefs.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.uic.edu/'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = HTMLrefs()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    p.close()
    print 'all html links :'
    p.ShowRefs()
```

Constructor and ShowRefs

of the class HTMLrefs

We now use a list as object data attribute.

```
def __init__(self):
    """
    Initializes the list of links.
    """
    HTMLParser.__init__(self)
    self.refs = []

def ShowRefs(self):
    """
    Prints the HTML refs to screen.
    """
    for each in self.refs:
        print each
```

Constructor and ShowRefs

of the class HTMLrefs

We now use a list as object data attribute.

```
def __init__(self):
    """
    Initializes the list of links.
    """
    HTMLParser.__init__(self)
    self.refs = []

def ShowRefs(self):
    """
    Prints the HTML refs to screen.
    """
    for each in self.refs:
        print each
```

Constructor and ShowRefs

of the class HTMLrefs

We now use a list as object data attribute.

```
def __init__(self):
    """
    Initializes the list of links.
    """
    HTMLParser.__init__(self)
    self.refs = []

def ShowRefs(self):
    """
    Prints the HTML refs to screen.
    """
    for each in self.refs:
        print each
```

Filtering Attributes

lambda expression, filter, and list comprehension

Attributes are lists of tuples: `[(.,.),(.,.),...]`

e.g.: `[('href', 'learning.shtml'), ...]`
→ link is the y in a `(x,y)` tuple

```
def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'a' and
    stores links for href attributes.
    """
    if tag == 'a':
        f = lambda (x,y): x == 'href'
        F = filter(f,attrs)
        L = [ y for (x,y) in F ]
        self.refs = self.refs + L
```

Filtering Attributes

lambda expression, filter, and list comprehension

Attributes are lists of tuples: `[(.,.),(.,.),...]`

e.g.: `[('href', 'learning.shtml'), ...]`
→ link is the y in a `(x,y)` tuple

```
def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'a' and
    stores links for href attributes.
    """
    if tag == 'a':
        f = lambda (x,y): x == 'href'
        F = filter(f,attrs)
        L = [ y for (x,y) in F ]
        self.refs = self.refs + L
```

Filtering Attributes

lambda expression, filter, and list comprehension

Attributes are lists of tuples: `[(.,.),(.,.),...]`

e.g.: `[('href', 'learning.shtml'), ...]`
→ link is the y in a `(x,y)` tuple

```
def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'a' and
    stores links for href attributes.
    """
    if tag == 'a':
        f = lambda (x,y): x == 'href'
        F = filter(f,attrs)
        L = [ y for (x,y) in F ]
        self.refs = self.refs + L
```

Filtering Attributes

lambda expression, filter, and list comprehension

Attributes are lists of tuples: `[(.,.),(.,.),...]`

e.g.: `[('href', 'learning.shtml'), ...]`
→ link is the y in a `(x,y)` tuple

```
def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'a' and
    stores links for href attributes.
    """
    if tag == 'a':
        f = lambda (x,y): x == 'href'
        F = filter(f,attrs)
        L = [ y for (x,y) in F ]
        self.refs = self.refs + L
```

Parsing HTML

Selecting pdf Files

Automatic download of files of type `.pdf`.

Steps in the script:

1. look for tags of type `'a'`
2. filter the attributes
   $\rightarrow$ file name in second element of tuple
3. only retain names with `.pdf` extension

Selecting pdf Files

Automatic download of files of type `.pdf`.

Steps in the script:

1. look for tags of type `'a'`
2. filter the attributes
   → file name in second element of tuple
3. only retain names with `.pdf` extension

Selecting pdf Files

Automatic download of files of type `.pdf`.

Steps in the script:

1. look for tags of type `'a'`
2. filter the attributes
   $\rightarrow$ file name in second element of tuple
3. only retain names with `.pdf` extension

the Class pdfFiles

```
from HTMLParser import HTMLParser
from urllib import urlopen

class pdfFiles(HTMLParser):
    """
    Scans attributes of 'a' tags for .pdf files.
    """
    def __init__(self):
        """
        Initializes the list of .pdf files.
        """
    def handle_starttag(self, tag, attrs):
        """
        For tags equal to 'a' looks for
        attributes ending in .pdf.
        """
    def ShowFiles(self):
        """
        Prints the list of files to screen.
        """
```

the Class pdfFiles

```
from HTMLParser import HTMLParser
from urllib import urlopen

class pdfFiles(HTMLParser):
    """
    Scans attributes of 'a' tags for .pdf files.
    """
    def __init__(self):
        """
        Initializes the list of .pdf files.
        """
    def handle_starttag(self, tag, attrs):
        """
        For tags equal to 'a' looks for
        attributes ending in .pdf.
        """
    def ShowFiles(self):
        """
        Prints the list of files to screen.
        """
```

the Class pdfFiles

```
from HTMLParser import HTMLParser
from urllib import urlopen

class pdfFiles(HTMLParser):
    """
    Scans attributes of 'a' tags for .pdf files.
    """
    def __init__(self):
        """
        Initializes the list of .pdf files.
        """
    def handle_starttag(self, tag, attrs):
        """
        For tags equal to 'a' looks for
        attributes ending in .pdf.
        """
    def ShowFiles(self):
        """
        Prints the list of files to screen.
        """
```

the Function main( )

of the script pyclassfiles.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.math.uic.edu/~jan/mcs275/main.html'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = pdfFiles()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    print 'pdf files on ' + page + ' :'
    p.ShowFiles()
    print 'number of files :', len(p.pdfFiles)
```

the Function main( )

of the script pyclassfiles.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.math.uic.edu/~jan/mcs275/main.html'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = pdfFiles()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    print 'pdf files on ' + page + ' :'
    p.ShowFiles()
    print 'number of files :', len(p.pdfFiles)
```

the Function main( )

of the script pyclassfiles.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://www.math.uic.edu/~jan/mcs275/main.html'
    print 'opening %s ...' % page
    f = urlopen(page)
    p = pdfFiles()
    while True:
        data = f.read(80)
        if data == '': break
        p.feed(data)
    print 'pdf files on ' + page + ' :'
    p.ShowFiles()
    print 'number of files :', len(p.pdfFiles)
```

Constructor and ShowFiles

of the class pdfFiles

```
def __init__(self):
    """
    Initializes the list of .pdf files.
    """
    HTMLParser.__init__(self)
    self.pdfFiles = []
```

We will sort the files before printing.

```
def ShowFiles(self):
    """
    Prints the list of files to screen.
    """
    self.pdfFiles.sort()
    for each in self.pdfFiles: print each
```

Constructor and ShowFiles

of the class pdfFiles

```
def __init__(self):
    """
    Initializes the list of .pdf files.
    """
    HTMLParser.__init__(self)
    self.pdfFiles = []
```

We will sort the files before printing.

```
def ShowFiles(self):
    """
    Prints the list of files to screen.
    """
    self.pdfFiles.sort()
    for each in self.pdfFiles: print each
```

Constructor and ShowFiles

of the class pdfFiles

```
def __init__(self):
    """
    Initializes the list of .pdf files.
    """
    HTMLParser.__init__(self)
    self.pdfFiles = []
```

We will sort the files before printing.

```
def ShowFiles(self):
    """
    Prints the list of files to screen.
    """
    self.pdfFiles.sort()
    for each in self.pdfFiles: print each
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`

No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`
No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`
No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`
No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`
No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Filtering Attributes

lambda expression, filter, and list comprehension

Look for `[('href', 'webcrawl.pdf')]`
No crash when name is less than 4 characters.

```
def handle_starttag(self, tag, attrs):
    """
    For tags equal to 'a' looks for
    attributes ending in .pdf.
    """
    if tag == 'a':
        A = [ y for (x,y) in attrs ]
        L = filter(lambda x: len(x)>3, A)
        F = filter(lambda x: x[-4:] == '.pdf', L)
        self.pdfFiles = self.pdfFiles + F
```

Getting Text formatted in Bold

looking for important data

Goal: extract text between and .

The important data is often emphasized in bold,
similar for titles between headers,
or for data in tables.

Method: use switch as state variable:

1. encounter : turn switch on
   → action done by `handle_starttag`
2. encounter : turn switch off
   → action done by `handle_endtag`

As long as switch is on, collect data.

Getting Text formatted in Bold

looking for important data

Goal: extract text between and .

The important data is often emphasized in bold,
similar for titles between headers,
or for data in tables.

Method: use switch as state variable:

1. encounter : turn switch on
   → action done by `handle_starttag`
2. encounter : turn switch off
   → action done by `handle_endtag`

As long as switch is on, collect data.

Getting Text formatted in Bold

looking for important data

Goal: extract text between and .

The important data is often emphasized in bold,
similar for titles between headers,
or for data in tables.

Method: use switch as state variable:

1. encounter : turn switch on
   → action done by `handle_starttag`
2. encounter : turn switch off
   → action done by `handle_endtag`

As long as switch is on, collect data.

Getting Text formatted in Bold

looking for important data

Goal: extract text between and .

The important data is often emphasized in bold,
similar for titles between headers,
or for data in tables.

Method: use switch as state variable:

1. encounter : turn switch on
   → action done by `handle_starttag`
2. encounter : turn switch off
   → action done by `handle_endtag`

As long as switch is on, collect data.

Getting Text formatted in Bold

looking for important data

Goal: extract text between and .

The important data is often emphasized in bold,
similar for titles between headers,
or for data in tables.

Method: use switch as state variable:

1. encounter : turn switch on
   → action done by `handle_starttag`
2. encounter : turn switch off
   → action done by `handle_endtag`

As long as switch is on, collect data.

Parsing HTML

the Class BoldText

```
class BoldText(HTMLParser):
    """
    Exports a switch for bold text.
    """
    def __init__(self):
        """
        Initializes the switch to False.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'b'
        and flips on the isbold flag.
        """
    def handle_endtag(self, tag):
        """
        Looks for tags equal to 'b'
        and flips off the isbold flag.
        """
```

the Class BoldText

```
class BoldText(HTMLParser):
    """
    Exports a switch for bold text.
    """
    def __init__(self):
        """
        Initializes the switch to False.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'b'
        and flips on the isbold flag.
        """
    def handle_endtag(self, tag):
        """
        Looks for tags equal to 'b'
        and flips off the isbold flag.
        """
```

the Class BoldText

```
class BoldText(HTMLParser):
    """
    Exports a switch for bold text.
    """
    def __init__(self):
        """
        Initializes the switch to False.
        """
    def handle_starttag(self, tag, attrs):
        """
        Looks for tags equal to 'b'
        and flips on the isbold flag.
        """
    def handle_endtag(self, tag):
        """
        Looks for tags equal to 'b'
        and flips off the isbold flag.
        """
```

the Function main( )

of the class boldtext.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://docs.python.org/lib/module-HTMLParser.ht
    print 'opening %s ...' % page
    f = urlopen(page)
    p = BoldText()
    s = ''
    while True:
        data = f.read(1)
        if data == '': break
        p.feed(data)
        if p.isbold:
            s = s + data
        else:
            if s != '': print s
            s = ''
```

the Function main( )

of the class boldtext.py

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://docs.python.org/lib/module-HTMLParser.ht
    print 'opening %s ...' % page
    f = urlopen(page)
    p = BoldText()
    s = ''
    while True:
        data = f.read(1)
        if data == '': break
        p.feed(data)
        if p.isbold:
            s = s + data
        else:
            if s != '': print s
            s = ''
```

the Function `main()`

of the class `boldtext.py`

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://docs.python.org/lib/module-HTMLParser.ht
    print 'opening %s ...' % page
    f = urlopen(page)
    p = BoldText()
    s = ''
    while True:
        data = f.read(1)
        if data == '': break
        p.feed(data)
        if p.isbold:
            s = s + data
        else:
            if s != '': print s
            s = ''
```

the Function `main()`

of the class `boldtext.py`

```
def main():
    """
    Opens a web page and parses it.
    """
    page = 'http://docs.python.org/lib/module-HTMLParser.ht
    print 'opening %s ...' % page
    f = urlopen(page)
    p = BoldText()
    s = ''
    while True:
        data = f.read(1)
        if data == '': break
        p.feed(data)
        if p.isbold:
            s = s + data
        else:
            if s != '': print s
            s = ''
```

Implementing the Methods of the Class

```
def __init__(self):
    """
    Initializes the switch to False.
    """
    HTMLParser.__init__(self)
    self.isbold = False
def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'b'
    and flips on the isbold flag.
    """
    if tag == 'b': self.isbold = True
def handle_endtag(self, tag):
    """
    Looks for tags equal to 'b'
    and flips off the isbold flag.
    """
    if tag == 'b': self.isbold = False
```

Implementing the Methods of the Class

```
def __init__(self):
    """
    Initializes the switch to False.
    """
    HTMLParser.__init__(self)
    self.isbold = False

def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'b'
    and flips on the isbold flag.
    """
    if tag == 'b': self.isbold = True

def handle_endtag(self, tag):
    """
    Looks for tags equal to 'b'
    and flips off the isbold flag.
    """
    if tag == 'b': self.isbold = False
```

Implementing the Methods of the Class

```
def __init__(self):
    """
    Initializes the switch to False.
    """
    HTMLParser.__init__(self)
    self.isbold = False

def handle_starttag(self, tag, attrs):
    """
    Looks for tags equal to 'b'
    and flips on the isbold flag.
    """
    if tag == 'b': self.isbold = True

def handle_endtag(self, tag):
    """
    Looks for tags equal to 'b'
    and flips off the isbold flag.
    """
    if tag == 'b': self.isbold = False
```

Summary + Assignments

`HTMLParser` is convenient to parse HTML
$\rightarrow$ makes it easier to retrieve data from web
Assignments:

1. Write a script that prompts the user for an URL and that finds the number of forms on the web page. The script should not crash when the page fails to open, but it should then display an error message.
2. Modify `pdfclassfiles.py` so it looks for files with the extension `.py`.
3. Write a script to automatically download and print the last printer friendly version of the slides.
4. Consider `webcrawler.py` of lecture 31. Use `HTMLParser` to write a shorter version.
5. Change class `BoldText` so all text formatted in bold is stored in a list in an object data attribute.