Web Clients and Crawlers

Web Clients
- alternatives to web browsers
- opening a web page and copying its content

Scanning files
- looking for strings between double quotes
- parsing URLs for the server location

Web Crawlers
- making requests recursively
- incremental development, modular design of code

MCS 275 Lecture 31
Programming Tools and File Management
Jan Verschelde, 2 April 2008

Web Clients and Crawlers

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
→ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   → update most recent information
3. *crawl* the web: request recursively
   → operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
$\rightarrow$ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   $\rightarrow$ update most recent information
3. *crawl* the web: request recursively
   $\rightarrow$ operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
→ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   → update most recent information
3. *crawl* the web: request recursively
   → operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
$\rightarrow$ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   $\rightarrow$ update most recent information
3. *crawl* the web: request recursively
   $\rightarrow$ operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
$\rightarrow$ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   $\rightarrow$ update most recent information
3. *crawl* the web: request recursively
   $\rightarrow$ operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
$\rightarrow$ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   $\rightarrow$ update most recent information
3. *crawl* the web: request recursively
   $\rightarrow$ operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients

alternatives to web browsers

Recall testing `ourwebserver.py` last lecture.
$\rightarrow$ the client is a browser, e.g.: Netscape, Firefox, ...

But we can browse the web using scripts.

Why do we want to do this?

1. *more efficient*: no overhead from GUI
2. *in control*: request only what we need
   $\rightarrow$ update most recent information
3. *crawl* the web: request recursively
   $\rightarrow$ operate like a search engine

How?

use `urllib` and `urlparse` modules

Web Clients and Crawlers

Copying a Web Page to a File

using the urllib module

Syntax:

```
urllib.retrieve( < URL >, < file name > )
```

Example:

```
from urllib import retrieve
retrieve('http://www.python.org','wpt.html')
```

Opening a web page with `urllib.urlopen`:

```
from urllib import urlopen
< object like file > = urlopen( < URL > )
data = < object like file >.read( < size > )
< object like file >.close()
```

→ process web pages like we handle files

Copying a Web Page to a File

using the `urllib` module

Syntax:

```
urllib.retrieve( < URL >, < file name > )
```

Example:

```
from urllib import retrieve
retrieve('http://www.python.org','wpt.html')
```

Opening a web page with `urllib.urlopen`:

```
from urllib import urlopen
< object like file > = urlopen( < URL > )
data = < object like file >.read( < size > )
< object like file >.close()
```

→ process web pages like we handle files

Copying a Web Page to a File

using the `urllib` module

Syntax:

```
urllib.retrieve( < URL >, < file name > )
```

Example:

```
  from urllib import retrieve
  retrieve('http://www.python.org','wpt.html')
```

Opening a web page with `urllib.urlopen`:

```
  from urllib import urlopen
  < object like file > = urlopen( < URL > )
  data = < object like file >.read( < size > )
  < object like file >.close()
```

→ process web pages like we handle files

Copying a Web Page to a File

using the `urllib` module

Syntax:
```
urllib.retrieve( < URL >, < file name > )
```

Example:

```
from urllib import retrieve
retrieve('http://www.python.org','wpt.html')
```

Opening a web page with `urllib.urlopen`:

```
from urllib import urlopen
< object like file > = urlopen( < URL > )
data = < object like file >.read( < size > )
< object like file >.close()
```

→ process web pages like we handle files

Copying a Web Page to a File

using the `urllib` module

Syntax:

```
urllib.retrieve( < URL >, < file name > )
```

Example:

```
from urllib import retrieve
retrieve('http://www.python.org','wpt.html')
```

Opening a web page with `urllib.urlopen`:

```
from urllib import urlopen
< object like file > = urlopen( < URL > )
data = < object like file >.read( < size > )
< object like file >.close()
```

→ process web pages like we handle files

Function to copy a Web Page to a File

```
def copypage(url,file):
    """
    Given the URL for the web page,
    a copy of its contents is written to file.
    Both url and file are strings.
    """
    import urllib
    copyfile = open(file,'w')
    f = urllib.urlopen(url)
    while True:
        data = f.read(80)
        if data == '': break
        copyfile.write(data)
    f.close()
    copyfile.close()
```

Function to copy a Web Page to a File

```
def copypage(url,file):
    """
    Given the URL for the web page,
    a copy of its contents is written to file.
    Both url and file are strings.
    """
    import urllib
    copyfile = open(file,'w')
    f = urllib.urlopen(url)
    while True:
        data = f.read(80)
        if data == '': break
        copyfile.write(data)
    f.close()
    copyfile.close()
```

Function to copy a Web Page to a File

```
def copypage(url,file):
   """
   Given the URL for the web page,
   a copy of its contents is written to file.
   Both url and file are strings.
   """
   import urllib
   copyfile = open(file,'w')
   f = urllib.urlopen(url)
   while True:
      data = f.read(80)
      if data == '': break
      copyfile.write(data)
   f.close()
   copyfile.close()
```

Function to copy a Web Page to a File

```
def copypage(url,file):
    """
    Given the URL for the web page,
    a copy of its contents is written to file.
    Both url and file are strings.
    """
    import urllib
    copyfile = open(file,'w')
    f = urllib.urlopen(url)
    while True:
        data = f.read(80)
        if data == '': break
        copyfile.write(data)
    f.close()
    copyfile.close()
```

Function to copy a Web Page to a File

```
def copypage(url,file):
    """
    Given the URL for the web page,
    a copy of its contents is written to file.
    Both url and file are strings.
    """
    import urllib
    copyfile = open(file,'w')
    f = urllib.urlopen(url)
    while True:
        data = f.read(80)
        if data == '': break
        copyfile.write(data)
    f.close()
    copyfile.close()
```

Web Clients and Crawlers

Web Clients
- alternatives to web browsers
- opening a web page and copying its content

Scanning files
- looking for strings between double quotes
- parsing URLs for the server location

Web Crawlers
- making requests recursively
- incremental development, modular design of code

Scanning HTML Files

Applications to scan an HTML file:

1. search for particular information,
2. navigate to where the page refers to.

Example (1): download all `.py` files from
`http://www.math.uic.edu/~jan/mcs275/main.html`

Example (2): retrieve all URLs the page
`http://www.python.org` refers to.

What is common between these two examples:

`.py` *files and URLs appear between* `"` *and* `"`

→ scan for all strings between double quotes

Scanning HTML Files

Applications to scan an HTML file:

1. search for particular information,
2. navigate to where the page refers to.

Example (1): download all `.py` files from
`http://www.math.uic.edu/~jan/mcs275/main.html`

Example (2): retrieve all URLs the page
`http://www.python.org` refers to.

What is common between these two examples:

`.py` *files and URLs appear between* `"` *and* `"`

→ scan for all strings between double quotes

Scanning HTML Files

Applications to scan an HTML file:

1. search for particular information,
2. navigate to where the page refers to.

Example (1): download all `.py` files from `http://www.math.uic.edu/~jan/mcs275/main.html`

Example (2): retrieve all URLs the page `http://www.python.org` refers to.

What is common between these two examples:

> `.py` *files and URLs appear between* `"` *and* `"`

$\rightarrow$ scan for all strings between double quotes

Scanning HTML Files

Applications to scan an HTML file:

1. search for particular information,
2. navigate to where the page refers to.

Example (1): download all `.py` files from `http://www.math.uic.edu/~jan/mcs275/main.html`

Example (2): retrieve all URLs the page `http://www.python.org` refers to.

What is common between these two examples:

`.py` *files and URLs appear between* `"` *and* `"`

$\rightarrow$ scan for all strings between double quotes

Scanning HTML Files

Applications to scan an HTML file:

1. search for particular information,
2. navigate to where the page refers to.

Example (1): download all `.py` files from
`http://www.math.uic.edu/~jan/mcs275/main.html`

Example (2): retrieve all URLs the page
`http://www.python.org` refers to.

What is common between these two examples:

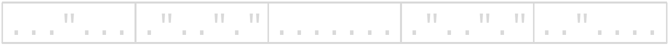
`.py` *files and URLs appear between* `"` *and* `"`

$\rightarrow$ scan for all strings between double quotes

Problem Statement
scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

$\rightarrow$ Two functions:

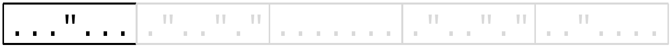
1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

→ Two functions:

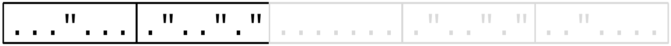
1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

$\rightarrow$ Two functions:

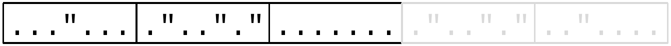
1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

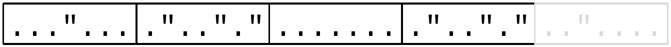
→ Two functions:

1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement
scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

$\rightarrow$ Two functions:

1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input: a file, or object like a file.
Output: list of all strings between double quotes.

Recall that we read files with fixed size buffer:

. . . " . . .	. " . . " . "	. . . .	. " . . " . "	. . " . . . .

For double quoted strings which run across two buffers we need another buffer.

Two buffers: one for reading strings from file,
one for buffering double quoted string.

→ Two functions:

1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input: a file, or object like a file.
Output: list of all strings between double quotes.

Recall that we read files with fixed size buffer:

. . . " . . .	. " . . " . "	. . . .	. " . . " . "	. . " . . . .

For double quoted strings which run across two buffers we need another buffer.

Two buffers: one for reading strings from file,
one for buffering double quoted string.

→ Two functions:

1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:

..."...	."..".".	....	."..".".	.."....

For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

→ Two functions:

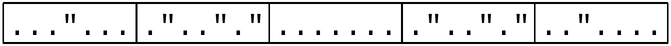
1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers:	one for reading strings from file,
	one for buffering double quoted string.

$\rightarrow$ Two functions:

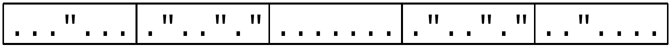
1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Problem Statement

scanning for double quoted strings

Input:	a file, or object like a file.
Output:	list of all strings between double quotes.

Recall that we read files with fixed size buffer:



For double quoted strings which run across two buffers we need another buffer.

Two buffers: one for reading strings from file,
one for buffering double quoted string.

$\rightarrow$ Two functions:

1. read buffered data from file,
2. scan the data buffer for double quoted strings.

Reading Strings from File

```
def QuotedStrings(file):
   """
   Given a file object, this function scans
   the file and returns a list of all strings
   on the file enclosed between double quotes.
   """
   L = []
   buffer = ''
   while True:
      data = file.read(80)
      if data == '': break
      (L,buffer) = UpdateQstrings(L,buffer,data)
   return L
```

Reading Strings from File

```
def QuotedStrings(file):
   """
   Given a file object, this function scans
   the file and returns a list of all strings
   on the file enclosed between double quotes.
   """
   L = []
   buffer = ''
   while True:
      data = file.read(80)
      if data == '': break
      (L,buffer) = UpdateQstrings(L,buffer,data)
   return L
```

Reading Strings from File

```
def QuotedStrings(file):
   """
   Given a file object, this function scans
   the file and returns a list of all strings
   on the file enclosed between double quotes.
   """
   L = []
   buffer = ''
   while True:
      data = file.read(80)
      if data == '': break
      (L,buffer) = UpdateQstrings(L,buffer,data)
   return L
```

Reading Strings from File

```
def QuotedStrings(file):
   """
   Given a file object, this function scans
   the file and returns a list of all strings
   on the file enclosed between double quotes.
   """
   L = []
   buffer = ''
   while True:
      data = file.read(80)
      if data == '': break
      (L,buffer) = UpdateQstrings(L,buffer,data)
   return L
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...     L = [], b = '"...'
."..".".    L = ['....','.'], b = ''
.......     L = ['....','.'], b = ''
."..".".    L = ['....','.','..'], b = '"'
.."....     L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = '"...'
."..".".     L = ['....','.'], b = ''
.......      L = ['....','.'], b = ''
."..".".     L = ['....','.','..'], b = '"'
.."....      L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
    """
    L is a list of double quoted strings,
    b buffers a double quoted string, and
    s is the data string to be processed.
    Returns an update of (L,b).
    """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = '"...'
.".."."      L = ['....','.'], b = ''
.......      L = ['....','.'], b = ''
.".."."      L = ['....','.','..'], b = '"'
.."....      L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
    """
    L is a list of double quoted strings,
    b buffers a double quoted string, and
    s is the data string to be processed.
    Returns an update of (L,b).
    """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

`..."...`	`L = [], b = '"...'`
`."..".".`	`L = ['....','.'], b = ''`
`.......`	`L = ['....','.'], b = ''`
`."..".".`	`L = ['....','.','..'], b = '"'`
`.."....`	`L = ['....','.','..','..'], b = ''`

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = '"...'
.".."."      L = ['....','.'], b = ''
.......      L = ['....','.'], b = ''
.".."."      L = ['....','.','..'], b = '"'
.."....      L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
    """
    L is a list of double quoted strings,
    b buffers a double quoted string, and
    s is the data string to be processed.
    Returns an update of (L,b).
    """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = '"...'
."..".".     L = ['....','.'], b = ''
.......      L = ['....','.'], b = ''
."..".".     L = ['....','.','..'], b = '"'
.."....      L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...     L = [], b = '"...'
."..".".    L = ['....','.'], b = ''
.......     L = ['....','.'], b = ''
."..".".    L = ['....','.','..'], b = '"'
.."....     L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = '"...'
."..".."     L = ['....','.'], b = ''
.......      L = ['....','.'], b = ''
."..".."     L = ['....','.','..'], b = '"'
.."....      L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
    """
    L is a list of double quoted strings,
    b buffers a double quoted string, and
    s is the data string to be processed.
    Returns an update of (L,b).
    """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...      L = [], b = ’"...’
."..".".     L = [’....’,’.’], b = ”
.......      L = [’....’,’.’], b = ”
."..".".     L = [’....’,’.’,’..’], b = ’"’
.."....      L = [’....’,’.’,’..’,’..’], b = ”
```

```
def UpdateQstrings(L,b,s):
    """
    L is a list of double quoted strings,
    b buffers a double quoted string, and
    s is the data string to be processed.
    Returns an update of (L,b).
    """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...     L = [], b = '"...'
."..".".    L = ['....','.'], b = ''
.......     L = ['....','.'], b = ''
."..".".    L = ['....','.','..'], b = '"'
.."....     L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Processing the Buffers

In L we store the double quoted strings.
In b we buffer the double quoted strings.

```
..."...    L = [], b = '"...'
."..".\"    L = ['....','.'], b = ''
.......    L = ['....','.'], b = ''
."..".\"    L = ['....','.','..'], b = '"'
.."....    L = ['....','.','..','..'], b = ''
```

```
def UpdateQstrings(L,b,s):
   """
   L is a list of double quoted strings,
   b buffers a double quoted string, and
   s is the data string to be processed.
   Returns an update of (L,b).
   """
```

Code for UpdateQstrings

```
def UpdateQstrings(L,b,s):
   "..."
   nb = b
   for i in range(0,len(s)):
      if nb == '':
         if s[i] == '\"':
            nb = 'o' # 'o' is for 'opened'
      else:
         if s[i] != '\"':
            nb += s[i]
         else:         # do not store 'o'
            L.append(nb[1:len(nb)])
            nb = ''
   return (L,nb)
```

Code for UpdateQstrings

```
def UpdateQstrings(L,b,s):
   ".."
   nb = b
   for i in range(0,len(s)):
      if nb == '':
         if s[i] == '\"':
            nb = 'o' # 'o' is for 'opened'
      else:
         if s[i] != '\"':
            nb += s[i]
         else:        # do not store 'o'
            L.append(nb[1:len(nb)])
            nb = ''
   return (L,nb)
```

Code for UpdateQstrings

```
def UpdateQstrings(L,b,s):
   ".."
   nb = b
   for i in range(0,len(s)):
      if nb == '':
         if s[i] == '\"':
            nb = 'o' # 'o' is for 'opened'
      else:
         if s[i] != '\"':
            nb += s[i]
         else:        # do not store 'o'
            L.append(nb[1:len(nb)])
            nb = ''
   return (L,nb)
```

Code for UpdateQstrings

```
def UpdateQstrings(L,b,s):
   ".."
   nb = b
   for i in range(0,len(s)):
      if nb == '':
         if s[i] == '\"':
            nb = 'o' # 'o' is for 'opened'
      else:
         if s[i] != '\"':
            nb += s[i]
         else:        # do not store 'o'
            L.append(nb[1:len(nb)])
            nb = ''
   return (L,nb)
```

Code for UpdateQstrings

```
def UpdateQstrings(L,b,s):
   ".."
   nb = b
   for i in range(0,len(s)):
      if nb == '':
         if s[i] == '\"':
            nb = 'o' # 'o' is for 'opened'
      else:
         if s[i] != '\"':
            nb += s[i]
         else:        # do not store 'o'
            L.append(nb[1:len(nb)])
            nb = ''
   return (L,nb)
```

The Function main()

```
def main():
    """
    Prompts the user for a file name and
    scans the file for double quoted strings.
    """
    print 'getting double quoted strings'
    name = raw_input('Give file name : ')
    file = open(name,'r')
    L = QuotedStrings(file)
    print L
    file.close()

if __name__=="__main__": main()
```

The Function main()

```
def main():
   """
   Prompts the user for a file name and
   scans the file for double quoted strings.
   """
   print 'getting double quoted strings'
   name = raw_input('Give file name : ')
   file = open(name,'r')
   L = QuotedStrings(file)
   print L
   file.close()

if __name__=="__main__": main()
```

Web Clients and Crawlers

Scanning Web Pages for URLs

Recall the second example application:
list all URLs referred to at `http://www.python.org`

```
def main():
    """
    Prompts the user for a web page,
    and prints all URLs this page refers to.
    """
    print 'listing reachable locations'
    page = raw_input('Give URL : ')
    L = HTTPlinks(page)
    print 'found %d HTTP links' % len(L)
    ShowLocations(L)
```

Scanning Web Pages for URLs

Recall the second example application:
list all URLs referred to at `http://www.python.org`

```
def main():
   """
   Prompts the user for a web page,
   and prints all URLs this page refers to.
   """
   print 'listing reachable locations'
   page = raw_input('Give URL : ')
   L = HTTPlinks(page)
   print 'found %d HTTP links' % len(L)
   ShowLocations(L)
```

Scanning Web Pages for URLs

Recall the second example application:
list all URLs referred to at `http://www.python.org`

```
def main():
    """
    Prompts the user for a web page,
    and prints all URLs this page refers to.
    """
    print 'listing reachable locations'
    page = raw_input('Give URL : ')
    L = HTTPlinks(page)
    print 'found %d HTTP links' % len(L)
    ShowLocations(L)
```

Filtering double quoted String

```
from scanquotes import UpdateQstrings

def HTTPfilter(L):
   """
   Returns from the list L only those strings
   which begin with http.
   """
   H = []
   for s in L:
      if len(s) > 4:
         if s[0:4] == 'http': H.append(s)
   return H
```

Filtering double quoted String

```
from scanquotes import UpdateQstrings

def HTTPfilter(L):
   """
   Returns from the list L only those strings
   which begin with http.
   """
   H = []
   for s in L:
      if len(s) > 4:
         if s[0:4] == 'http': H.append(s)
   return H
```

Scanning HTML File for HTTP Strings

```
def HTTPlinks(url):
   """
   Given the URL for the web page,
   returns the list of all HTTP strings.
   """
   import urllib
   f = urllib.urlopen(url)
   L = []; b = ''
   while True:
      data = f.read(80)
      if data == '': break
      (L,b) = UpdateQstrings(L,b,data)
      L = HTTPfilter(L)
   f.close()
   return L
```

Scanning HTML File for HTTP Strings

```
def HTTPlinks(url):
   """
   Given the URL for the web page,
   returns the list of all HTTP strings.
   """
   import urllib
   f = urllib.urlopen(url)
   L = []; b = ''
   while True:
      data = f.read(80)
      if data == '': break
      (L,b) = UpdateQstrings(L,b,data)
      L = HTTPfilter(L)
   f.close()
   return L
```

Scanning HTML File for HTTP Strings

```
def HTTPlinks(url):
    """
    Given the URL for the web page,
    returns the list of all HTTP strings.
    """
    import urllib
    f = urllib.urlopen(url)
    L = []; b = ''
    while True:
        data = f.read(80)
        if data == '': break
        (L,b) = UpdateQstrings(L,b,data)
        L = HTTPfilter(L)
    f.close()
    return L
```

Scanning HTML File for HTTP Strings

```
def HTTPlinks(url):
    """
    Given the URL for the web page,
    returns the list of all HTTP strings.
    """
    import urllib
    f = urllib.urlopen(url)
    L = []; b = ''
    while True:
        data = f.read(80)
        if data == '': break
        (L,b) = UpdateQstrings(L,b,data)
        L = HTTPfilter(L)
    f.close()
    return L
```

Showing only Server Locations

using the module `urlparse`

An URL consists of 6 parts

```
protocol://location/path:parameters?query#frag
```

Given URL u, `urlparse.urlparse`(u) returns 6-tuple.

```
def ShowLocations(L):
   """
   Shows the locations of the URL in L.
   """
   from urlparse import urlparse
   for h in L:
      p = urlparse(h)
      print p[1]
```

Showing only Server Locations
using the module `urlparse`

An URL consists of 6 parts

```
protocol://location/path:parameters?query#frag
```

Given URL u, `urlparse.urlparse`(u) returns 6-tuple.

```
def ShowLocations(L):
   """
   Shows the locations of the URL in L.
   """
   from urlparse import urlparse
   for h in L:
      p = urlparse(h)
      print p[1]
```

Web Clients and Crawlers

Web Clients

alternatives to web browsers

opening a web page and copying its content

Scanning files

looking for strings between double quotes

parsing URLs for the server location

Web Crawlers

making requests recursively

incremental development, modular design of code

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
    3.1 open the web page defined by location of URL,
    3.2 compute the list of all URLs on that page.

→ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   3.1 open the web page defined by location of URL,
   3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
    3.1 open the web page defined by location of URL,
    3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
   - 3.1 open the web page defined by location of URL,
   - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Crawlers

making requests recursively

Scanning HTML files and browsing:

1. given a URL, open a web page,
2. compute the list of all URLs in the page,
3. for all URLs in the list do:
    - 3.1 open the web page defined by location of URL,
    - 3.2 compute the list of all URLs on that page.

$\rightarrow$ continue recursively, *crawling* the web

Things to consider:

1. remove duplicates from list of URLs,
2. do not turn back to pages visited before,
3. limit the levels of recursion,
4. some links will not work.

Similar to finding a path in a maze, but now we are interested in all intermediate nodes along the path.

Web Clients and Crawlers

Web Clients

- alternatives to web browsers
- opening a web page and copying its content

Scanning files

- looking for strings between double quotes
- parsing URLs for the server location

Web Crawlers

- making requests recursively
- incremental development, modular design of code

Modular Design of Web Crawler

use what we have developed so far

```
# L-31 MCS 275 Wed 2 Apr 2008 : webcrawler.py

# Prompts the user for a URL and the maximal
# depth of the recursion tree.
# Lists all locations of web servers that can
# reached starting from the user given URL.

from scanquotes import UpdateQstrings
from scanhttplinks import HTTPfilter, HTTPlinks
```

Still left to write:

1. management of list of server locations,
2. recursive function to crawl the web.

Modular Design of Web Crawler

use what we have developed so far

```
# L-31 MCS 275 Wed 2 Apr 2008 : webcrawler.py

# Prompts the user for a URL and the maximal
# depth of the recursion tree.
# Lists all locations of web servers that can
# reached starting from the user given URL.

from scanquotes import UpdateQstrings
from scanhttplinks import HTTPfilter, HTTPlinks
```

Still left to write:

1. management of list of server locations,
2. recursive function to crawl the web.

Modular Design of Web Crawler

use what we have developed so far

```
# L-31 MCS 275 Wed 2 Apr 2008 : webcrawler.py

# Prompts the user for a URL and the maximal
# depth of the recursion tree.
# Lists all locations of web servers that can
# reached starting from the user given URL.

from scanquotes import UpdateQstrings
from scanhttplinks import HTTPfilter, HTTPlinks
```

Still left to write:

1. management of list of server locations,
2. recursive function to crawl the web.

Retain only new Locations

```
def NewLocations(L,V):
    """
    Given the list L of new URLs and the
    list of already visited locations,
    returns the list of new locations,
    locations not yet visited earlier.
    """
    from urlparse import urlparse
    newL = []
    for h in L:
        p = urlparse(h)
        loc = p[1]
        if not loc in newL:
            if not loc in V:
                newL.append(loc)
    return newL
```

Retain only new Locations

```
def NewLocations(L,V):
    """
    Given the list L of new URLs and the
    list of already visited locations,
    returns the list of new locations,
    locations not yet visited earlier.
    """
    from urlparse import urlparse
    newL = []
    for h in L:
        p = urlparse(h)
        loc = p[1]
        if not loc in newL:
            if not loc in V:
                newL.append(loc)
    return newL
```

Retain only new Locations

```
def NewLocations(L,V):
    """
    Given the list L of new URLs and the
    list of already visited locations,
    returns the list of new locations,
    locations not yet visited earlier.
    """
    from urlparse import urlparse
    newL = []
    for h in L:
        p = urlparse(h)
        loc = p[1]
        if not loc in newL:
            if not loc in V:
                newL.append(loc)
    return newL
```

Retain only new Locations

```
def NewLocations(L,V):
    """
    Given the list L of new URLs and the
    list of already visited locations,
    returns the list of new locations,
    locations not yet visited earlier.
    """
    from urlparse import urlparse
    newL = []
    for h in L:
        p = urlparse(h)
        loc = p[1]
        if not loc in newL:
            if not loc in V:
                newL.append(loc)
    return newL
```

Retain only new Locations

```
def NewLocations(L,V):
    """
    Given the list L of new URLs and the
    list of already visited locations,
    returns the list of new locations,
    locations not yet visited earlier.
    """
    from urlparse import urlparse
    newL = []
    for h in L:
        p = urlparse(h)
        loc = p[1]
        if not loc in newL:
            if not loc in V:
                newL.append(loc)
    return newL
```

Some Links will not work...

Add an exception handler:

```
def HTTPlinks(url):
   """
   Given the URL for the web page,
   returns the list of all HTTP strings.
   """
   import urllib
   try:
      print 'opening ' + url + ' ...'
      f = urllib.urlopen(url)
   except:
      print 'opening ' + url + ' failed'
      return []
```

Some Links will not work...

Add an exception handler:

```
def HTTPlinks(url):
   """
   Given the URL for the web page,
   returns the list of all HTTP strings.
   """
   import urllib
   try:
      print 'opening ' + url + ' ...'
      f = urllib.urlopen(url)
   except:
      print 'opening ' + url + ' failed'
      return []
```

Unparsing URLs

using the `urlparse` module again

Recall that we only store the server locations.

To open a web page we also need to specify the protocol.

We apply `urlparse.urlunparse`

```
>>> from urlparse import urlunparse
>>> urlunparse(('http','www.python.org',
... '','','',''))
'http://www.python.org'
```

We must provide a 6-tuple as argument ...

Unparsing URLs

using the `urlparse` module again

Recall that we only store the server locations.

To open a web page we also need to specify the protocol.

We apply `urlparse.urlunparse`

```
>>> from urlparse import urlunparse
>>> urlunparse(('http','www.python.org',
... '','','',''))
'http://www.python.org'
```

We must provide a 6-tuple as argument ...

Unparsing URLs
using the `urlparse` module again

Recall that we only store the server locations.

To open a web page we also need to specify the protocol.

We apply `urlparse.urlunparse`

```
>>> from urlparse import urlunparse
>>> urlunparse(('http','www.python.org',
... '','','',''))
'http://www.python.org'
```

We must provide a 6-tuple as argument ...

Unparsing URLs

using the `urlparse` module again

Recall that we only store the server locations.

To open a web page we also need to specify the protocol.

We apply `urlparse.urlunparse`

```
>>> from urlparse import urlunparse
>>> urlunparse(('http','www.python.org',
... '','','',''))
'http://www.python.org'
```

We must provide a 6-tuple as argument ...

Running the Crawler

```
$ python webcrawler.py
crawling the web ...
Give URL : http://www.uic.edu
give maximal depth : 2
opening http://www.uic.edu ...
opening http://www.w3.org ...
opening http://www.csail.mit.edu ...
opening http://www.ercim.org ...
opening http://jigsaw.w3.org ...
opening http://validator.w3.org ...
opening http://www2008.org ...
opening http://www.bicc.com.cn ...
opening http://www.primelife.eu ...
```

.. it takes a while ..

```
total #locations : 538
```

Running the Crawler

```
$ python webcrawler.py
crawling the web ...
Give URL : http://www.uic.edu
give maximal depth : 2
opening http://www.uic.edu ...
opening http://www.w3.org ...
opening http://www.csail.mit.edu ...
opening http://www.ercim.org ...
opening http://jigsaw.w3.org ...
opening http://validator.w3.org ...
opening http://www2008.org ...
opening http://www.bicc.com.cn ...
opening http://www.primelife.eu ...
```

.. it takes a while ..

```
total #locations : 538
```

Running the Crawler

```
$ python webcrawler.py
crawling the web ...
Give URL : http://www.uic.edu
give maximal depth : 2
opening http://www.uic.edu ...
opening http://www.w3.org ...
opening http://www.csail.mit.edu ...
opening http://www.ercim.org ...
opening http://jigsaw.w3.org ...
opening http://validator.w3.org ...
opening http://www2008.org ...
opening http://www.bicc.com.cn ...
opening http://www.primelife.eu ...
```

.. it takes a while ..

```
total #locations : 538
```

The Function main()

```
def main():
   """
   Prompts the user for a web page,
   and prints all URLs this page refers to.
   """
   print 'crawling the web ...'
   page = raw_input('Give URL : ')
   k = input('give maximal depth : ')
   L = crawler(page,k,[])
   print 'reachable locations :', L
   print 'total #locations :', len(L)

if __name__=="__main__": main()
```

The Function main()

```
def main():
   """
   Prompts the user for a web page,
   and prints all URLs this page refers to.
   """
   print 'crawling the web ...'
   page = raw_input('Give URL : ')
   k = input('give maximal depth : ')
   L = crawler(page,k,[])
   print 'reachable locations :', L
   print 'total #locations :', len(L)

if __name__=="__main__": main()
```

Code for the Crawler

```
def crawler(url,k,V):
   """
   Returns the list V updated with the
   list of locations reachable from the
   given url using k steps.
   """
   from urlparse import urlunparse
   L = HTTPlinks(url)
   newL = NewLocations(L,V)
   newV = V + newL
   if k == 0:
      return newV
   else:
      for loc in newL:
         u = urlunparse(('http',loc,'','','',''))
         newV = crawler(u,k-1,newV)
      return newV
```

Code for the Crawler

```
def crawler(url,k,V):
    """
    Returns the list V updated with the
    list of locations reachable from the
    given url using k steps.
    """
    from urlparse import urlunparse
    L = HTTPlinks(url)
    newL = NewLocations(L,V)
    newV = V + newL
    if k == 0:
        return newV
    else:
        for loc in newL:
            u = urlunparse(('http',loc,'','','',''))
            newV = crawler(u,k-1,newV)
        return newV
```

Code for the Crawler

```
def crawler(url,k,V):
   """
   Returns the list V updated with the
   list of locations reachable from the
   given url using k steps.
   """
   from urlparse import urlunparse
   L = HTTPlinks(url)
   newL = NewLocations(L,V)
   newV = V + newL
   if k == 0:
      return newV
   else:
      for loc in newL:
         u = urlunparse(('http',loc,'','','',''))
         newV = crawler(u,k-1,newV)
      return newV
```

Code for the Crawler

```
def crawler(url,k,V):
    """
    Returns the list V updated with the
    list of locations reachable from the
    given url using k steps.
    """
    from urlparse import urlunparse
    L = HTTPlinks(url)
    newL = NewLocations(L,V)
    newV = V + newL
    if k == 0:
        return newV
    else:
        for loc in newL:
            u = urlunparse(('http',loc,'','','',''))
            newV = crawler(u,k-1,newV)
        return newV
```

Summary + Assignments

We covered more of chapter 14 in *Making Use of Python*.

Assignments:

1. Write script to download all `.py` files from `http://www.math.uic.edu/~jan/mcs275/main.html`
2. Limit the search of the crawler so that it only opens pages within the same domain. For example, if we start at a location ending with `edu`, we only open pages with locations ending with `edu`.
3. Adjust `webcrawler.py` to search for a path between two locations. The user is prompted for two URLs. Crawling stops if a path has been found.

Assignments collected on Friday 4 April:
#1 of L-26, #2 of L-27, #4 of L-28, #1 of L-29, #2 of L-30.