COMP10001 Foundations of Computing
The Internet and HTML

Semester 1, 2019

Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— version: 1449, date: March 1, 2019 —

Lecture Agenda

- Last lecture:
  - Advanced Lecture
- This lecture:
  - Project 2 review
  - Finishing recursion
  - Internet
  - HTML

Lecture Outline

`index` - Binary Search

- Input: sorted `list` of numbers
- Output: the index of a given number `x`, or `None` if it's not in the list
- Thinking recursively and cleverly (`n=len(lst)`):

$$index(x, lst) = \begin{cases} None & \text{if lst is empty} \\ n/2 & \text{if lst[n/2] is x} \\ index(x, lst[:n/2]) & \text{if x} < \text{lst[n/2]} \\ n/2 + index(x, lst[n/2:]) & \text{otherwise} \end{cases}$$

0	1	2	3	4	5	6	7
1	3	10	12	15	45	86	91

Binary Search: Recursive Solution

```
def bsearch(val,nlist):
    return bs_rec(val,nlist,0,len(nlist)-1)

def bs_rec(val,nlist,start,end):
    if start > end:
        return None
    mid = start+(end-start)//2
    if nlist[mid] == val:
        return mid
    elif nlist[mid] < val:
        return bs_rec(val,nlist,mid+1,end)
    else:
        return bs_rec(val,nlist,start,mid-1)
```


Binary Search: Iterative Solution

... but again, there's an equally elegant iterative solution:

```
def bs_it(val,nlist):
    start = 0
    end = len(nlist) - 1
    while start < end:
        mid = start+(end-start)//2
        if nlist[mid] == val:
            return mid
        elif nlist[mid] < val:
            start = mid + 1
        else:
            end = mid - 1
    return None
```

So When *Should* You Use Recursion?

Recursion comes to its fore when an iterative solution would involve a level of iterative nesting proportionate to the size of the input, e.g.:

- the powerset problem: given a list of items, return the list of unique groupings of those items (each in the form of a list)
- the change problem: given a list of different currency denominations (e.g. `[5,10,20,50,100,200]`), calculate the number of distinct ways of forming a given amount of money from those denominations

Making Head and Tail of Recursion

- Recursion occurs in two basic forms:
  1. **head recursion:** recurse first, then perform some local calculation

```
def counter_head(n):
    if n < 0: return
    counter_head(n-1)
    print n
```

  2. **tail recursion:** perform some local calculation, then recurse

```
def counter_tail(n):
    if n < 0: return
    print n
    counter_tail(n-1)
```


Recursion: A Final Word

- Recursion is very powerful, and should always be used with caution:
  - function calls are expensive, meaning deep recursion comes at a price
  - always make sure to catch the base case, and avoid infinite recursion!
  - there is often a more efficient iterative solution to the problem, although there may not be a general iterative solution (esp. in cases where the obvious solution involves arbitrary levels of nested iteration)
  - recursion is elegant, but elegance $\neq$ more readable or efficient

Lecture Outline

The Internet: Brief History

- 1950s–1960s:
  - Linking computers and terminals
  - Local Area Networks (LANs)
- 1970s–1980s:
  - Linking multiple LANs (government and defence)
  - Wide Area Networks (WANs)
  - Packet switching for efficiency and robustness
  - TCP/IP for global connectivity
- 1990s–2000s (and beyond):
  - The killer app: World Wide Web (WWW)
  - Web 2.0

Addressing Machines: IPs

- Each device on the Internet has a unique "IP address"
- In IPv4, IP addresses are represented as 4 "8-bit" integers, each in the range $[0, 255]$, e.g. 128.250.36.33 is Tim's main web server
- IP addresses can be allocated to a device either "statically" (an IP is reserved for a given device) or "dynamically" (an IP is allocated to a device dynamically when it connects to the Internt)
- There is increasing momentum to move to IPv6 ($8\times$4-digit "hexadecimal" numbers) because we are rapidly running out of IP addresses

Addressing Machines: Hostnames

- Humans tend to find sequences of numbers hard to remember, so devices also tend to have "hostnames" such as `hum.csse.unimelb.edu.au` made up of (case-insensitive) letters and full stops
- Hostnames are structured hierarchically relative to a domain name (e.g. `unimelb.edu.au`) and end with a "top-level domain" (TLD, e.g. `au`)
- Hostnames resolve to IP addresses via the Domain Name System (DNS) using "name servers"

URLs

- Internet resources are "addressed" via URLs ("Uniform Resource Locators"):
  `http://nlp.stanford.edu:8080/parser/`
  `ftp://ftp.unimelb.edu.au/pub/www/ughb-book2007.tar.gz`
  `mms://www.microsoft.com/videos/a_streaming_video.wmv`
- URLs are made up of the following parts:

  scheme://hostname:port/path

  where:
  - scheme = the "protocol" for accessing the file
  - hostname = the device the file lives on
  - port = access port (optional)
  - path = where the file lives on the device

Lecture Outline

Applications and Ports

- Multiple applications run over IP networks:
  - Email
  - World Wide Web (HTTP)
  - FTP
  - Chat/Instant Messaging
  - Video Streaming
- Machines communicate with other machines over the Internet via a collection of numbered "ports", differentiated by application:
  - 21 = FTP
  - 25 = outgoing email (SMTP)
  - 80 = HTTP
  - 110 = incoming email (POP3)

URL: Examples

- `http://nlp.stanford.edu:8080/parser/`
  - protocol = HTTP
  - hostname = `nlp.stanford.edu`
  - port = 8080
  - path = `parser`
- So what happens when I type `google.com` into my browser?
  - protocol = HTTP (client-side default)
  - hostname = `www.google.com` (client-side default)
  - port = 80 (by default from protocol)
  - path = `index.html` (or similar; server-side default)

HTML: Introduction

- The primary language used to "mark up" web documents is HTML ("Hypertext Markup Language"); we will focus on HTML5
- HTML is made up of "elements" ("tags" and "entities"), which are used to mark up "content"
- HTML tags are enclosed in "angle brackets" (e.g. `<tag>`), and take the form of: (1) "empty elements" (e.g. `<tag/>`; note backslash at *end* of tag), or (2) tag pairs (e.g. `<tag></tag>`; note backslash at *start* of closing tag)
- HTML tags may optionally contain "attributes"

HTML: Mark-up Basics

- Given some (textual) content:

```
How much wood could a woodchuck chuck
```

we mark up regions with tag pairs, e.g.:

```
How much <b>wood</b> could a <i>wood</i>chuck chuck
```

which renders as:

How much **wood** could a *wood*chuck chuck

- The basic textual tag pairs are:
  - `<i></i>`: *italics*
  - `<b></b>`: **bold**
  - `<u></u>`: <u>underline</u>

HTML: Document Structure

- All HTML documents should start with a declaration of "document type" on the first line:

  `<!DOCTYPE html>`

  be enclosed within `<html></html>` tags, and contain a "head" (`<head></head>`) and "body" (`<body></body`) respectively, i.e.:

```
<!DOCTYPE html>
<html>
<head>
...
</head>
<body>
...
</body>
</html>
```


HTML: The Body

- Common elements of the body of an HTML document to structure the text are:
  - headers (`<h1></h1>`, `<h2></h2>`, ...
  - paragraphs (`<p></p>`)
  - line breaks (`<br/>`)
  - horizontal lines (`<hr/>`)
- "Hyperlinks" can be inserted with `<a href=""></a>` over "anchor text"

```
<body>
<a href="./index.html">Recursive link!</a>
</body>
```


HTML: Stacking up Mark-up

- It is possible to stack up mark-up, but tags have to be closed in the reverse order of opening (a la a "stack"), i.e. must be nested within one another, e.g.

```
How much <u><b>wood</b></u> could a <i>wood</i>chuck chuck
```

which renders as:

How much **<u>wood</u>** could a *wood*chuck chuck

HTML: The Head

- The head of an HTML document standardly contains a title:

  `<title></title>`

  and will also often contain "meta-data" as attributes to empty `<meta/>` elements, including keywords, character encoding information, a description of the site, ...

```
<head>
<title>HTML Introduction</title>
<meta name="description" content="An intro to HTML"/>
<meta name="keywords" content="HTML,computing,coolness"/>
<meta name="author" content="Tim Baldwin"/>
<meta charset="UTF-8"/>
</head>
```


HTML: White Space

- White space can be inserted for readability, but is largely ignored by the browser: the browser turns any sequence of white space characters into a single space before processing
- Exception: preformatted information between `<pre></pre>` tags is displayed as it appears

HTML: More on Hyperlinking

- URLs in hyperlinks can be:
  - "absolute URLs", i.e. complete URLs including hostname, such as `http://server/directory_path/filename`
  - "relative URLs", i.e. relative to the current location on the same server, such as `./a_file_in_the_same_directory.html`
- In relative URLs, we often use two special characters:
  - `.` = Current directory
  - `..` = Parent directory (one level up)
- Relative URLs are more flexible, as it is possible to move web page sets around as a group without having to update URLs

HTML: Multimedia Content

- Images (of varying formats) can be included with `<img src=""alt=""/>`, where `src` specifies the image file location, and `alt` is alternate text (if the image doesn't load)
- Audio files can be included with `<audio><source src=""type=""/>alt</audio>`
- Video files can be included with `<video><source src=""type=""/>alt</video>`

HTML: Lists

- Enclose unnumbered (bulleted) lists with `<ul></ul>`
  - declare list items with `<li></li>`
- Enclose ordered (numbered) lists with `<ol></ol>`
  - declare list items with `<li></li>`

```
<ul>
    <li>Paul</li>
    <li>John</li>
    <li>George</li>
    <li>Ringo</li>
</ul>
<ol>
    <li>Computing</li>
    <li>Everything else</li>
</ol>
```


HTML: Tables

- Enclose tables with `<table></table>` (optional `border` attribute)
- Render table a row at a time, enclosing each row with `<tr></tr>`, and each cell with `<td></td>` or `<th></th>` (for column headers)

```
<table border="1">
    <tr>
        <th>First name</th>
        <th>Last name</th>
    </tr>
    <tr>
        <td>Nic</td>
        <td>Geard</td>
    </tr>
</table>
```


HTML: Tables

- Enclose tables with `<table></table>` (with optional `border` attribute)
- Render the table a row at a time, enclosing each row with `<tr></tr>`, and each cell with `<td></td>` or `<th></th>` (for column headers)

First name	Last name
Nic	Geard

HTML: Entities

- HTML "entities" are special characters, which take the form `&entity;`
- The most commonly used entities are:

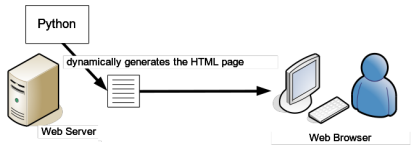
`&quot;`	"	` `	space
`<`	<	`&apos;`	'
`>`	>	`&`	&

- There are also entities for characters with diacritics, such as ü = `&uuml;`, é = `&eacute;`, ì = `&igrave;`

Serving HTML Pages

- Static HTML

- Dynamic HTML using Python within Grok



So How does it Work?

- Dynamically generating a web page from within Grok simply involves saving the HTML output of a Python script to a file

Exercise

- Write a Python function list2html that accepts a required argument list, which contains a list of values in string type each. list2html must return a string containing an HTML-encoded table suitable for embedding into an HTML document, presenting the data. For example: ['a'] should become

```
<table><tr><td>a</td></tr></table>
```


Solution

```
def list2html(mylist):
    out = "<table>"
    for item in mylist:
        out += "<tr>"
        out += "<td>{}</td>".format(item)
        out += "</tr>"
    return(out + "</table>")
```


Lecture Summary

- What are IPs and hostnames, and what is their role?
- What are ports?
- What are URLs and how are they structured?
- What are HTML elements, tags, attributes and entities?
- What are the essential elements of an HTML document?
- How do you include hyperlinks/multimedia files in HTML documents?
- How do you typeset lists and tables?
- How to generate dynamic HTML pages from Python