CS6
Practical System Skills
Fall 2019 edition
Leonhard Spiegelberg
lspiegel@cs.brown.edu
11.99 Recap

Last lecture: Regular expressions + use in `grep/sed/awk`

- match one arbitrary character
  * 0 or more matches of preceding item
  + 1 or more matches of preceding item
  ? 0 or 1 match of preceding item (optional)
  (...) capturing group
  {m} match exactly m times
  {m,n} match at least m, at most n times
  [...] define character class
Recap Quiz - let's develop a regex!

<table>
<thead>
<tr>
<th>Regex task</th>
<th>Examples</th>
<th>should not match</th>
</tr>
</thead>
<tbody>
<tr>
<td>match hexadecimal RGB colors</td>
<td>#123456</td>
<td>123456</td>
</tr>
<tr>
<td></td>
<td>#abDE87</td>
<td>#XYabcd</td>
</tr>
<tr>
<td></td>
<td>#001200</td>
<td>#ghabcd</td>
</tr>
<tr>
<td>match non-negative integers 0-999 without leading zeros</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>02</td>
</tr>
<tr>
<td></td>
<td>98</td>
<td>003</td>
</tr>
<tr>
<td></td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

⇒ please collaborate & use e.g. regex101.com!
Recap Quiz - let's develop a regex!

<table>
<thead>
<tr>
<th>Regex task</th>
<th>Solution (not unique per se)</th>
</tr>
</thead>
<tbody>
<tr>
<td>match hexademical RGB colors</td>
<td>#[0-9a-fA-F]{6}</td>
</tr>
<tr>
<td>integers 0-999 without leading zeros</td>
<td>^([0-9]</td>
</tr>
</tbody>
</table>

⇒ please collaborate & use e.g. regex101.com!
⇒ BRE or ERE syntax is both fine. (Use ERE, it's easier…)
Another regex exercise

Can you develop a regex to match numbers 0-255?

yes! break up into ranges. ⇒ Useful e.g. for IPv4 addresses

Solution:
^([0-9]|1[0-9][0-9]|1[0-9]{0,2}|25[0-5]|2[0-4][0-9][0-9])$  

⇒ Good idea to have semantic logic in regex?
Practical example: regex

Fetching lecture slides via grep/sed/curl:

curl -s https://cs.brown.edu/courses/csci0060/lectures.html | grep -Eo 'href=\".*\.pdf\"' | sed 's/href=\"//g' | sed 's/\^https://cs.brown.edu/courses/csci0060/' | xargs -n1 curl -O

Try to fetch lecture slides from other courses:
E.g., https://web.stanford.edu/class/cs142/, http://cs.brown.edu/courses/cs1951g/,...
12.00 Resources

⇒ There are many resources to learn web development online, e.g.
   https://www.w3schools.com/
   https://www.codecademy.com/learn/learn-html
   https://education.github.com/pack

⇒ Slides on HTML/CSS are based on Jon Duckett's HTML&CSS
   → Chapter 1-7
   → pp. 193-194 (escape characters in HTML)
   → Chapter 10-12 and Chapter 17
12.01 What happens when you access a webpage?

1. sends DNS request with URL
   - DNS Server

2. answers with IP address of server behind URL

3. HTTP GET request to retrieve HTML page
   - Web server

4. response, delivers HTML page

5. After parsing HTML page, request additional resources like stylesheets, scripts, images, videos, files, ...

6. responses

It depends a bit on the concrete webpage what happens!
12.02 HTTP requests

⇒ base of accessing a page and resources are HTTP requests, full documentation available under [https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html](https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html)

⇒ a HTTP request is a message send via TCP over port 80.

⇒ There are multiple versions:

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>1.0</td>
</tr>
<tr>
<td>1997</td>
<td>1.1</td>
</tr>
<tr>
<td>2015</td>
<td>2.0</td>
</tr>
<tr>
<td>2018</td>
<td>3.0</td>
</tr>
</tbody>
</table>

HTTP/3 will be coming 2019!
### HTTP GET request for course website

<table>
<thead>
<tr>
<th>HTTP GET request for course website</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /courses/csci0060/index.html HTTP/1.1</td>
</tr>
</tbody>
</table>

- Host: cs.brown.edu
- Connection: keep-alive
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36
- Accept: text/html
- Accept-Encoding: gzip, deflate, br
- Accept-Language: en-US, en

- First specifies type (GET), resource (/courses/csci0060/index.html) and protocol version

- Request header fields

- Empty line, after it an optional message body may be included
General structure of a HTTP/1.1 request

request message consists of

1. request line (e.g. GET / HTTP/1.1)
2. (optional) request header fields
3. empty line
4. (optional) message body

⇒ what happens after a request is sent?
   → The server responds (hopefully)!
12.02 HTTP responses

⇒ The server to which the request was sent to, answers with a response. Similar to a request the response message consists of:

1. status line including *status code* and *status message*  
   e.g. **HTTP/1.1. 200 OK**
2. response header fields
3. empty line
4. optional message body

<table>
<thead>
<tr>
<th>Example response message</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP/1.1 200 OK</strong></td>
</tr>
<tr>
<td>Server: nginx/1.10.3</td>
</tr>
<tr>
<td>Date: Thu, 10 Oct 2019 13:04:41 GMT</td>
</tr>
<tr>
<td>Content-Type: text/html</td>
</tr>
<tr>
<td>Content-Length: 2084</td>
</tr>
<tr>
<td>Connection: keep-alive</td>
</tr>
<tr>
<td>Last-Modified: Tue, 08 Oct 2019 20:41:58 GMT</td>
</tr>
<tr>
<td>ETag: &quot;15fa-5946c3317148d-gzip&quot;</td>
</tr>
<tr>
<td>Accept-Ranges: bytes</td>
</tr>
<tr>
<td>Vary: Accept-Encoding</td>
</tr>
<tr>
<td>Content-Encoding: gzip</td>
</tr>
</tbody>
</table>
## 12.02 HTTP requests

There are multiple kinds of HTTP requests:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>read request, i.e. retrieve specified resource. Should not have side effects.</td>
</tr>
<tr>
<td><strong>HEAD</strong></td>
<td>similar to GET, but only delivers headers back. I.e. no message body (data is not delivered)</td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>write request, i.e. post data to server to store/process...</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>store data under supplied URI, i.e. for file upload</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>delete data under supplied URI, i.e. for file deletion</td>
</tr>
</tbody>
</table>
12.03 Diving into the web

⇒ You can use Chrome → Help → Developer → Developer Tools to access tools to trace how a webpage is loaded.

⇒ Perform a hard refresh to see resources loaded/requests in the network header.

⇒ DEMO
12.03 Curl

⇒ unpractical to write requests by hand, thus tool to make life easier:
   → curl (or wget) to perform requests for you!

⇒ cURL is a widespread tool to interact with URLs.
   → https://curl.haxx.se/

⇒ can be used to download files using multiple protocols, but also to issue HTTP requests!
   → many options to set headers, message, request type, ...
12.03 Curl - GET request

curl performs per default a GET request

→ detailed documentation under https://curl.haxx.se/docs/httpscripting.html
→ use -v to understand what curl does, or --trace-ascii logfile.txt

There are several options to perform HEAD or POST requests:

<table>
<thead>
<tr>
<th>Option</th>
<th>--head</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-I</td>
<td></td>
<td>performs HEAD request</td>
</tr>
<tr>
<td>-d data</td>
<td>--data data</td>
<td>performs POST request with data as message body</td>
</tr>
<tr>
<td>-L</td>
<td>--location</td>
<td>sometimes servers redirect pages (i.e. you'll receive a 3XX response), use this option to let curl follow the redirect.</td>
</tr>
</tbody>
</table>
12.03 More on cURL

⇒ you can also explicitly specify HTTP request details in cURL, e.g. via

curl --data "" --header "Accept: text/html" --request GET cs.brown.edu

⇒ for now: curl to retrieve web pages / resources

⇒ Later in CS6: Filling out web forms & testing REST APIs

Note:
Sometimes curl doesn't deliver something, because the webserver will only reveal a page to a web browser. To remedy this, add necessary headers to your request! E.g.,
https://developers.whatismybrowser.com/useragents/explore/ to find a user agent.
12.03 Curl, saving output to files

⇒ curl write output per default to stdout
   → i.e., could use stdout redirection to save file.

⇒ per default curl shows download progress, disable via \(-s\)

⇒ use \(-o\) to use last part of url as filename to save to.

⇒ Alternative: use \(-o\) filename to specify a filename to save to
   \((-o \; -\; will\; write\; to\; stdout)\).

Example:

curl \(-L\) \texttt{cs.brown.edu/courses/csci0060/assets/slides/slides9.pdf} \(-o\) lecture09.pdf
Writing a webpage from scratch...
12.04 HTML

HTML = Hypertext Markup Language

hypertext: text which contains links to other hypertext documents

markup: annotations to structure content

⇒ HTML is the format to ship webpages.

⇒ HTML pages are text documents.

⇒ There are multiple versions of HTML, in this course: HTML5
12.04 HTML tags

⇒ HTML consists of a series of (nested) elements.

⇒ HTML elements tell the browser how to display the content.

⇒ HTML elements are represented by tags.

⇒ HTML tags usually come in pairs with an opening tag `<tag>` and a closing tag `</tag>`, the content is enclosed by these two tags.

**Example:**

```html
<p>Some text here... </p>
```
12.04 HTML tags - continued

Some tags don't have a closing tag, their syntax is simply `<tag>`.

⇒ In HTML5 they are known as void elements (don't try to close them with `</tag>!`), as they must not contain any content.

⇒ tags may have attributes, syntax is:

```
<tag attribute="value"></tag>
```

Example (tag with several attributes):

```
<div class="blue" id="main" width="100px"></div>
```
12.04 HTML tags

⇒ HTML tag names and attribute names are case-insensitive but attribute values (and content) are case sensitive.

→ convention: lower case tag names and attribute names

Example:
<p>some text</p> is the same as <P>some text</P>
but <p>SOME text</p> is different than <p>some text</p>.

<div class="HELLO"></div> is the same as
<div CLASS="HELLO"></div> but different than
<div class="hello"></div>
12.04 HTML structure & comments

⇒ Structure of HTML document is defined by nesting tags, e.g.

<pre><code>&lt;outerTag&gt;
  &lt;innerTag&gt;
    text
  &lt;/innerTag&gt;
&lt;/outerTag&gt;
</code></pre>

⇒ comments can be added via <!-- comment goes here -->
12.04 Basic HTML webpage structure

```html
<!DOCTYPE html>
<html>
  <head>
    <title>This is the title of the webpage</title>
  </head>
  <body>
    <h1>This is the body of the page</h1>
    <p>Some content here…</p>
  </body>
</html>
```

⇒ save as file with html or htm extension and can open it via a web browser!
12.04 Tools for HTML development

⇒ a text editor is sufficient, however to be more productive you ideally want one with live reload functionality (i.e. displays changes without manual refresh).

⇒ There are many editors out there:
   E.g. brackets.io, Sublime Text, Atom.io, WebStorm, Adobe Dreamweaver, ...

⇒ Also many webtools provide live development with a live preview:
   codepen.io, playcode.io, scratchpad.io, ...
Welcome to the world of HTML tags...
1. Text manipulation tags
12.05 Text tags

⇒ Use h1, ..., h6 to define headings, e.g. <h1>Heading</h1>

⇒ <p></p> defines a paragraph, i.e. to put words into a paragraph enclose them with a p tag.

⇒ For changing the text format, the following tags can be used

  <b>bold text</b>  <!-- bold text -->
  <i>italic text</i>  <!-- italic text -->
  E=MC<sup>2</sup>  <!-- superscript -->
  H<sub>2</sub>O  <!-- subscript -->
12.05 line breaks & rules

⇒ use `<br>` to add a line break

⇒ `<hr>` will create a horizontal rule

⇒ both of these elements are void elements, as they do not contain any content and thus must not be closed!

**Note:** You'll often see `<br />` or `<hr />` too, the reason for this is because that this syntax is XHTML (an older HTML version) compatible. If you're only targeting HTML5, use `<br>` and `<hr>`.
12.05 Semantic text tags

⇒ **strong** is used to strongly emphasize content, *em* to emphasize content (by default displayed as bold or italic text).

⇒ `blockquote` can be used for a long quote, `q` for an inline quote (blockquote is per default displayed with an indent, `q` surrounds content with "").

⇒ `abbr` can be used to introduce an abbreviation, its title attribute is displayed when hovering over it.

**Example:**

```html
<abbr title="National Aeronautics and Space Administration">NASA</abbr>
```

⇒ There are many more semantic tags in HTML5 available.
2. Lists
12.06 lists

⇒ general structure to define a list to have one enclosing tag which specifies the list type, i.e.

- `ol` ordered list
- `ul` unordered list

and then items enclosed by a `li` (list item) tag.

Example:

```html
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```
3. Links
12.07 Anchor tags

 ⇒ Links are created via anchor tags <a></a>

 ⇒ can be used to link to another anchor tag within the same page or to another page

Basic syntax:

```html
<a href="http://cs.brown.edu">CS Department@Brown</a>
```


12.07 Anchor tags - linking to other pages

⇒ instead of fully qualified URLs, one can use absolute or relative URLs

⇒ Absolute URL: similar to absolute paths starts with /
   → / is the site root, not the root dir of the host!

⇒ relative URL: work like relative paths, e.g.

   `<a href="../music/top50.html">Billboard Top50</a>`

   links to page in `../music` folder.
12.07 Anchor tags - linking to the same page

⇒ Any element in HTML can have an id attribute assigned to

Example:

```html
<h1 id="first-main-heading">Heading</h1>
```

⇒ to link to a specific part of the page, reference the id with #, i.e.

```html
<a href="#first-main-heading">go to first heading</a>
```
12.07 Anchor tags - continued

⇒ <a href="mailto:lspiegel@cs.brown.edu">Email Leonhard</a>
starts user's email program
⇒ with target attribute, the destination where links are opened
is specified, i.e.

_blank: open link in new window
_self: open link in same frame it was clicked (default)
_parent: open link in the parent frame
4. Images
12.08 including images

⇒ images can be include using `<img>` tag (void element! no closing tag!)

⇒ typically, images are stored in a separate folder `img/` or `images/` at the site root.

⇒ General syntax:

```html
<img src="images/tux.png" alt="mascot of linux">
```
12.08 images continued

⇒ attributes height and width can be used to specify image display dimensions

→ you can uses different units to specify the value, e.g.
  \[
  \text{width=“100“ or width=“100px“ or width=“20\%“}
  \]

⇒ images can be aligned using the align attribute, e.g. when placing an image within a paragraph tag (\texttt{<p>}/\texttt{</p>}) with align attribute set, text flows around it.

→ use align="left" or align="right" for horizontal alignment
→ use align="top", align="middle" or align="bottom" for vertical.
5. Tables
12.09 HTML tables

⇒ use `<table></table>` to define a new table

⇒ a row is defined via `<tr></tr>`

→ to define individual table cells, use `<td></td>`

⇒ to define a header row, use instead of td, th tag.

⇒ with rowspan or colspan attribute, a table cell can span multiple rows or columns respectively
12.09 HTML tables

⇒ header row is usually enclosed by a thead tag
⇒ body using a tbody tag.
⇒ Many browser automatically add these tags
⇒ optionally, you may add a tfoot tag for a footer row.
⇒ There are tools to generate tables fast, e.g.

### 12.09 HTML table - example

```html
<table>
<thead>
  <tr>
    <th>country</th>
    <th>capital</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>U.S.A</td>
    <td>Washington, D.C.</td>
  </tr>
  <tr>
    <td>Canada</td>
    <td>Ottawa</td>
  </tr>
</tbody>
</table>
```
6. Structural tags
12.10 Structural tags

⇒ HTML5 introduced many new tags to structure the content of a website better (cf. Chapter 17, HTML&CSS)

⇒ use the `<nav>` tag together with a `<ul>` tag to define a navigation

⇒ parts of the website can be structured using the `<section>` tag

⇒ for a header/footer HTML5 provides a `<header>` and `<footer>` tag
CSS = Cascading Stylesheets
12.11 CSS

⇒ Cascading stylesheets (CSS) allow to change the appearance of a website, they're written in their own language.

⇒ Three options to use CSS:

1. write CSS code enclosed by `<style type="text/css"></style>` tag in head section

2. write CSS code in separate file and link it via a link tag within the webpage
   `<link href="css/style.css" type="text/css" rel="stylesheet">`

3. write CSS rules into style attribute, e.g.
   `<p style="color: #ccc; background-color: #000;">...</p>`
12.11 Basic CSS syntax

⇒ declarations are of the form `property: value`

⇒ the selector defines to which elements the declarations should be applied.
12.11 Basic CSS selectors

⇒ each element in HTML can have a class attribute.
→ used to apply a "style" (i.e. a CSS rule) to multiple elements
→ id attribute is used to uniquely identify an element

Full list of CSS selectors e.g. under [https://www.w3schools.com/cssref/css_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp).

tag{ ...}   apply rules to all <tag> elements
#name { ...} apply rules to all <tag id="name"> elements
.name { ...} apply rules to all <tag class="name"> elements
12.11 Basic CSS selectors

⇒ Selectors can be combined, e.g.

- `h1, h2, h3 {...}` apply rule to h1, h2 and h3 tags
- `p a {...}` apply rule to all `<a>` which are enclosed by `<p>` (can be multiple levels up)
- `p>a {...}` apply rule to all `<a>` which are children of `<p>`, i.e. `<p><a></a><p>` but not `<p><div><a></a></div></p>`
- `p.note` target only `p` tags which have class attribute `note`
12.11 Basic CSS selectors

\[ \text{p\#name} \] targets \text{p} tag which has \text{id} attribute with value \text{name}.

⇒ typically only one identifier is assigned to the id field, but one can assign multiple classes to the class attribute!

Example:

\[ \text{<p class="text text-justify">...</p>} \]
12.11 CSS properties

⇒ there are many properties (depending on tag type) that can be set.

⇒ A complete list can be found e.g. here [https://www.w3schools.com/cssref/](https://www.w3schools.com/cssref/)
12.11 CSS

⇒ Learning CSS requires some time
⇒ Many resources online available

Next lecture: More on CSS & practical web dev tricks!
13 Midterm

CS6 Practical System Skills
Fall 2019
Leonhard Spiegelberg lspiegel@cs.brown.edu
13.01 Logistics

⇒ 10/22 in CIT477, 4pm → counts 15%

⇒ You have 90min, however the exam will be designed for 60-70min.

⇒ paper based, if you need special accommodations please reach out!

⇒ you're allowed to bring one cheatsheet, i.e. one piece of US-letter with your own personal notes

⇒ questions are a mix of multiple choice, and part answers

→ you might be required to write some small scripts, however solutions should not exceed 10 lines. Often there's a one-liner.
Each cheatsheet must contain your **own** personal, handwritten notes.

It is ok to collaborate on the content of a cheatsheet, i.e. preparing together for a midterm.
⇒ Lectures 1-10 (today).

⇒ Focus on HOW to use commands, use your cheatheets for notes on options/commands.

⇒ Specific topics of interest:

  file paths, file permissions, streams, pipes, brace expansion, parameter expansion, arithmetic expansion, quoting, SSH + key setup, processes(especially signals), regular expressions & string processing, Basic HTML & everything else in homeworks + slides.
How to prepare?

⇒ Make sure you did all homeworks

⇒ make sure you can solve quizzes in recap sections.

⇒ write your cheatsheet

⇒ practice on permissions, regular expressions and piping commands

⇒ go to office hours & get help from your TAs!
Example question I

4. (2 points) You are in /home/ec2-user/data and running ls gives you

```
1  2  3  4  5  6  7  8  9 10 11 12 13 results
```

Each entry might refer to a file or a directory. You want to put all these entries (except results) into the folder results. Write a single command to achieve this (2P): If you don’t know how to achieve this with a single command, you may use up to 3 commands but you will only get 1P at most.

Solution: Can be done in multiple ways, e.g. mv {?,??,??} results/
Example question II

**Problem:** You're given the following output (stdout) from `curl cs.brown.edu/cs1951g`

Write a script (or command) to download all pdf files!

```html
<!DOCTYPE html> <html >
<body>
 <div id="maincontent">
  <h2>Slides</h2>
  <img src="images/puppets.gif" style="float:left;" />
  <p>The slides are posted after the material has been covered in class.</p>
  <ul>
   <li>(1/26) <a href="slides/00-Intro.pdf">00-Intro</a></li>
   <li>(2/2) <a href="slides/01-LPTheory.pdf">01-LPTheory</a> (complete)</li>
   <li>(2/9) <a href="slides/03-IPTTheory.pdf">03-IPTTheory</a> (complete)</li>
   <li>(2/23) <a href="slides/04-IndexFunds.pdf">04-IndexFunds</a></li>
   <li>(3/2) <a href="slides/05-ConvexTheory.pdf">05-ConvexTheory</a></li>
   <li>(3/2) <a href="slides/05b-DualityKKT.pdf">05b-DualityKKT</a></li>
   <li>(3/9) <a href="slides/06-Unconstrained.pdf">06-Unconstrained</a></li>
   <li>(3/16) <a href="slides/07-PortfolioOptimization.pdf">07-PortfolioOptimization</a></li>
  </ul>
 </div>
</body>
</html>
```
Example question II - solution

solution:

curl cs.brown.edu/cs1951g | grep -Eo "slides/.\.*\.pdf" | sed 's|^|cs.brown.edu/cs1951g/|' | xargs -n1 curl -O

<!DOCTYPE html> <html>
  <body>
    <div id="maincontent">
      <h2>Slides</h2>
      <img src="images/puppets.gif" style="float:left;" />
      <p>The slides are posted after the material has been covered in class.</p>
      <ul>
        <li>(1/26) <a href="slides/00-Intro.pdf">00-Intro</a></li>
        <li>(2/2) <a href="slides/01-LPTheory.pdf">01-LPTheory</a> (complete)</li>
        <li>(2/9) <a href="slides/03-IPTTheory.pdf">03-IPTTheory</a> (complete)</li>
        <li>(2/23) <a href="slides/04-IndexFunds.pdf">04-IndexFunds</a></li>
        <li>(3/2) <a href="slides/05-ConvexTheory.pdf">05-ConvexTheory</a></li>
        <li>(3/2) <a href="slides/05b-DualityKKT.pdf">05b-DualityKKT</a></li>
        <li>(3/9) <a href="slides/06-Unconstrained.pdf">06-Unconstrained</a></li>
        <li>(3/16) <a href="slides/07-PortfolioOptimization.pdf">07-PortfolioOptimization</a></li>
      </ul>
    </div>
  </body>
</html>
End of lecture.

Next class: Tue, 4pm-5:20pm @ CIT 477