# CSC 470 – Section 3

## Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 4

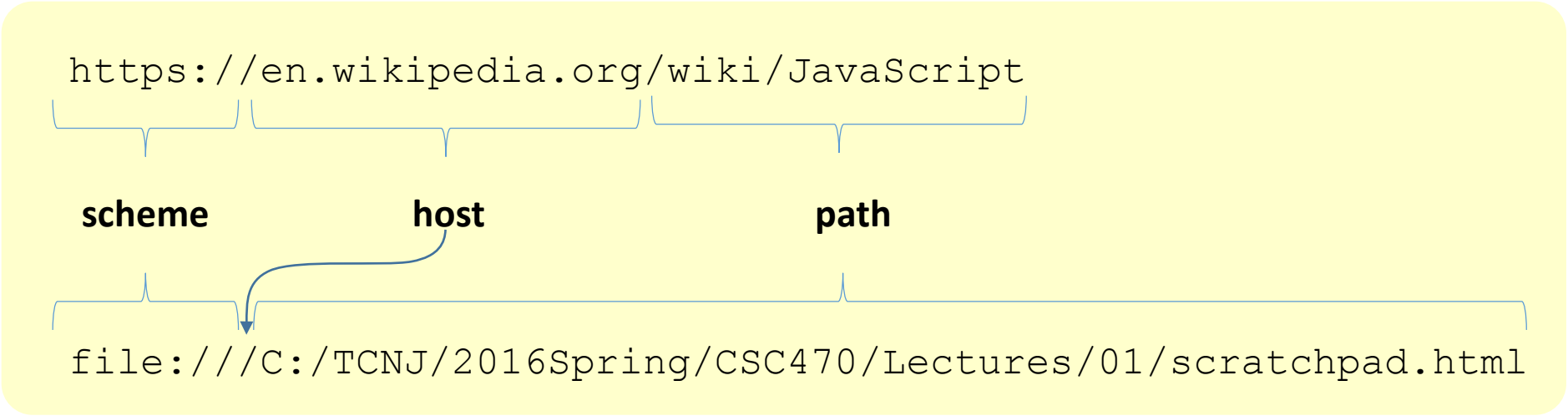
Eloquent JavaScript: Chapters 12, 13

# Web Browsers

- Web browsers were created as a simple document rendering and linking application
- A desire to make document interactive led to the introduction of JavaScript as an embedded scripting language
- Web browsers became the killer app for JavaScript
  - Without them, JavaScript would not have gained popularity
- Web browsers have been incrementally expanded to become a full-featured platform for client-side application development

# URL – A Simplified View

- Location of document to render is specified with a unique **Uniform Resource Locator (URL)**.
- URLs point to remote or local locations




**scheme** The means by which the resource should be accessed

- `http://` HyperText Transfer Protocol
- `file://` Local File System

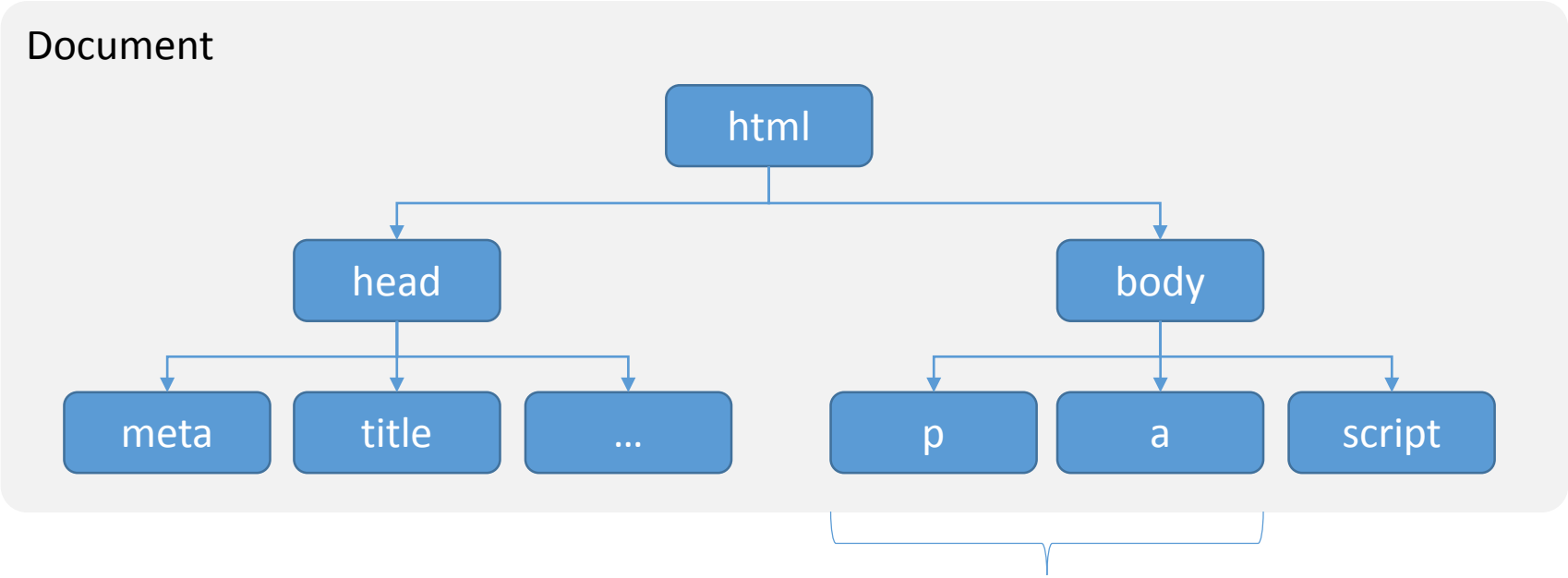
**server** The name or IP address of the machine on which the resource may be found

**path** The path on the machine where the resource is stored

**Note – These may be virtual – they may not have physical counterparts, such as machines or files*

# Document Object Model

- Rendered web pages are organized as an inverted tree of Node objects called the Document Object Model (DOM)




Only this part is rendered to the browser window

# HyperText Markup Language (HTML)

- DOM Nodes are specified in an HTML file as pairs of tags `<tag> … </tag>`
- Subordinate nodes are tag-pairs placed inside other tag pairs
- HTML is read when page loads, turned into DOM and rendered to window

*Sample HTML Document*

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Page Title</title>
  </head>
  <body>
    <p>Paragraph 1</p>
    <a href="http://tcnj.edu">TCNJ</a>
    <script type="text/javascript">
      // JavaScript Code Here
    </script>
  </body>
</html>
```


*Rendered HTML*

`<!doctype html>` is not an HTML tag, but a **Document Type Declaration** (DTD) indicating how the document that follows should be rendered

# HyperText Markup Language (HTML)

- Tags are delineated by angle brackets
- Tag pairs are start tags and end tags (/)
- Tag pairs may contain text or other tag pairs
- Empty tag pairs may be written as self-closing tag
- Tags may be parameterized with attributes

```
<tag> … </tag>
```

*start tag* *end tag*

```
<tag/>
```

```
attrname = "value"
```

tag name attribute name attribute value

```
<a href="http://tcnj.edu">TCNJ</a>
```

start tag tag contents end tag

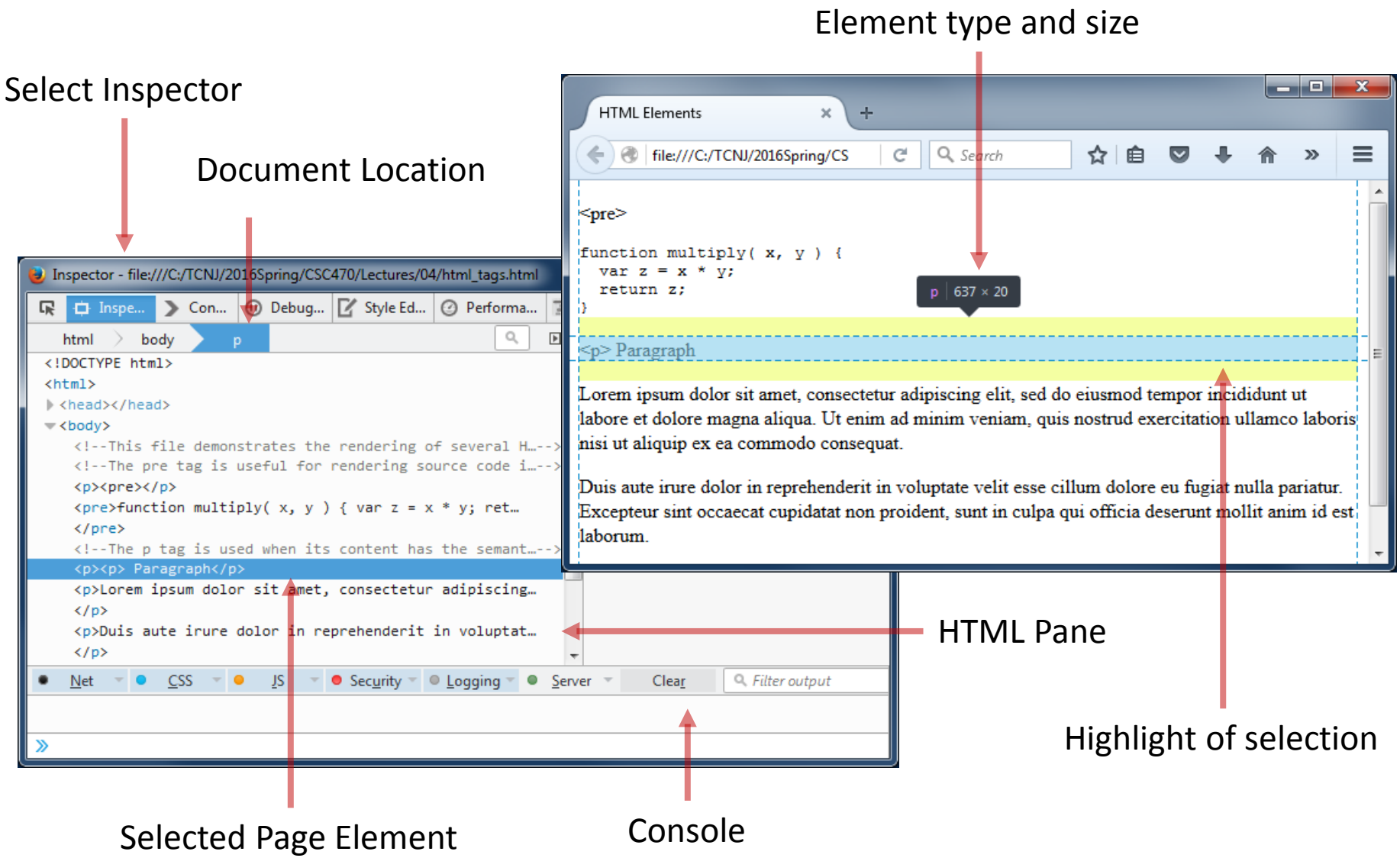
"a" is the anchor tag – used to render a hyperlink to the URL value of the "href" attribute

# Firefox Page Inspector

- The HTML and other aspects that define a web page may be visualized with the Firefox Inspector Tool
- Mouse over tag pairs highlights corresponding rendered part of web page in browser
- Examine and edit the HTML in the browser
- Inspect many other aspects that define the page, including
  - styles in effect
  - layout parameters such as location, size, margins and padding
  - fonts
  - event listeners
  - …

https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector

# Firefox Page Inspector



# Some Common HTML Tags: Block-Level

`<pre></pre>`

- Preformatted Text
  - Content is displayed in a fixed-width font and preserves both spaces and line breaks.
  - A block-level Element (vs. an inline-level element), creates page structure
    - Begins on a new line. Followed by vertical space.

`<p></p>`

- Paragraph Element
  - Lays out tag contents as a paragraph, with text wrapping
  - A block-level Element
  - A `<p>` should not be nested within another

`<div></div>`

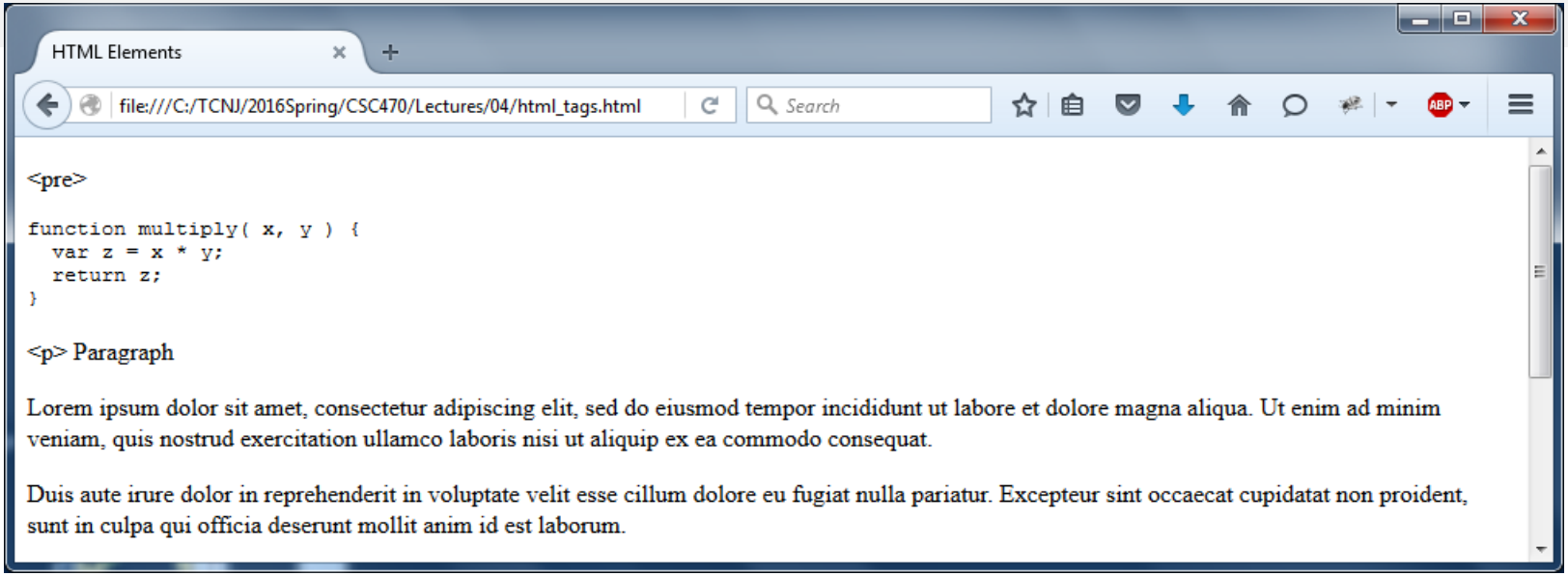
- Division Element
  - Generic block-level element
  - Used for blocks of content not meant to be paragraphs, semantically
  - `<div>` Elements may be nested, and are designed for that purpose

https://docs.webplatform.org/wiki/html/elements

04/html_tags.html

# Some Common HTML Tags: Block-Level

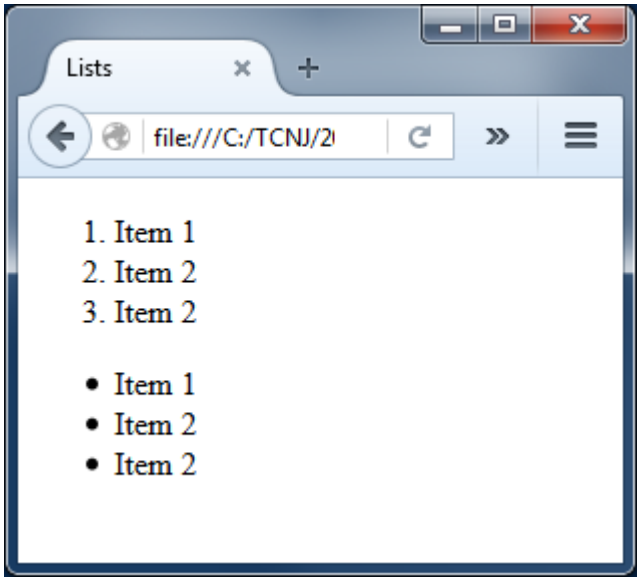
```
<!-- The pre tag is useful for rendering source code in a browser -->
    <p>&lt;pre&gt;</p>
    <pre>
function multiply( x, y ) {
  var z = x * y;
  return z;
}   </pre>

    <!-- The p tag is used when its content has the semantics of a document paragraph -->
    <p>&lt;p&gt; Paragraph</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
    et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
    aliquip ex ea commodo consequat.
    </p>

    <p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
    dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
    officia deserunt mollit anim id est laborum.
    </p>
```



# Some Common HTML Tags: Block-Level

- Two outline styles, ordered list `<ol></ol>` and unordered list `<ul></ul>`
- One or more list item tags appear with list tags as `<li></li>`

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Lists</title>
  </head>
  <body>

    <ol>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 2</li>
    </ol>

    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 2</li>
    </ul>

  </body>
</html>
```




https://docs.webplatform.org/wiki/html/elements

04/lists.html

# Some Common HTML Tags: Block-Level

- Rectangular tables are specified with the `<table></table>` tag
- Each row in a table is delineated with `<tr></tr>` tags
- Each cell in a table delineated with `<td></td>` tags

- Table rows may be grouped into three (optional) vertical blocks

  - Table Header `<thead></thead>`
  - Table Body `<tbody></tbody>`
  - Table Footer `<tfoot></tfoot>`

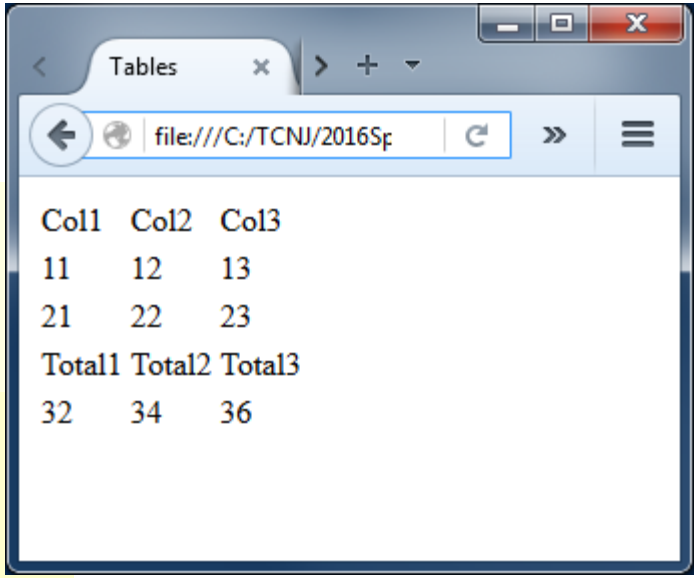
- Grouping is useful for styling rows independently (discuss later)

https://docs.webplatform.org/wiki/html/elements

04/tables.html

# Some Common HTML Tags: Block-Level

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Tables</title>
  </head>
  <body>

    <table>
      <thead>
        <tr>
          <td> Col1 </td><td> Col2 </td><td> Col3 </td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td> 11 </td><td> 12 </td><td> 13 </td>
        </tr>
        <tr>
          <td> 21 </td><td> 22 </td><td> 23 </td>
        </tr>
      </tbody>
      <tfoot>
        <tr>
          <td> Total1 </td><td> Total2 </td><td> Total3 </td>
        </tr>
        <tr>
          <td> 32 </td><td> 34 </td><td> 36 </td>
        </tr>
      </tfoot>
    </table>

  </body>
</html>
```




04/tables.html

# Some Common HTML Tags: Inline-Level

```
<a href="http://path/to/resource">Link Text</a>
```

- Anchor Tag
  - Renders a hyperlink

```
<img src="/path/to/image">
```

- Image
  - Embeds an image in a document
  - Does not have sub elements
  - Does not need to be closed

```
<!--   -->
```

- Comment
  - Content is not displayed or rendered
  - A comment starts with `<!--` and ends with `-->`
  - May contain any textual content
  - Comments do not nest

# Some Common HTML Tags

```
<!-- An a tag is an anchor and renders as a hyperlink that navigates to
a new URL when clicked -->

    <a href="http://tcnj.edu">TCNJ</a>

    <!-- An inline image -->
    <p>
      <img src="TCNJ.png">
    </p>
```

# Some Common HTML Tags: Objects

```
<button type="button">Button Text</button>
```

- Button
  - Renders a clickable element
  - Can place text or images inside a button element

```
<textarea>Text Content</textarea>
```

- Text Area
  - Renders a multi-line text input control

```
<form> </form>
```

- Form
  - An HTML form for user input, subsequently to be submitted to a website or service.
  - May contain multiple objects that accept user input

https://docs.webplatform.org/wiki/html/elements

04/html_tags.html

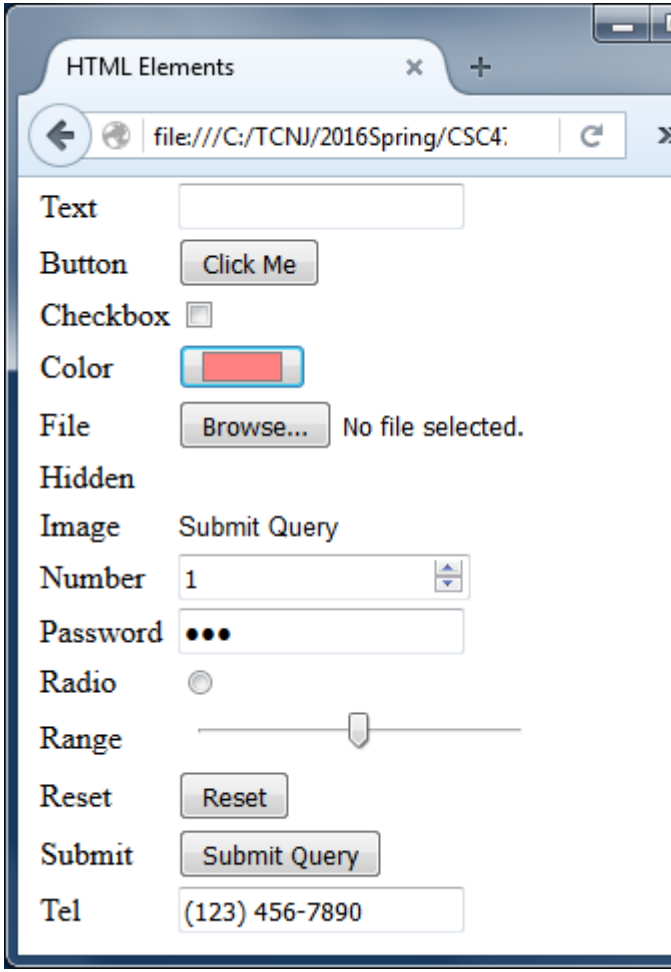
# Some Common HTML Tags: Objects

```
<input type="button" />
```

- A multipurpose element for form widgets.
- Widget type depends on `type` attribute value.

- Examples

| | |
|---|---|
| `type=text (default)` | `type=password` |
| `type=button` | `type=radio` |
| `type=checkbox` | `type=range` |
| `type=color` | `type=reset` |
| `type=file` | `type=submit` |
| `type=hidden` | `type=tel` |
| `type=image` | `type=number` |




04/html_tags.html

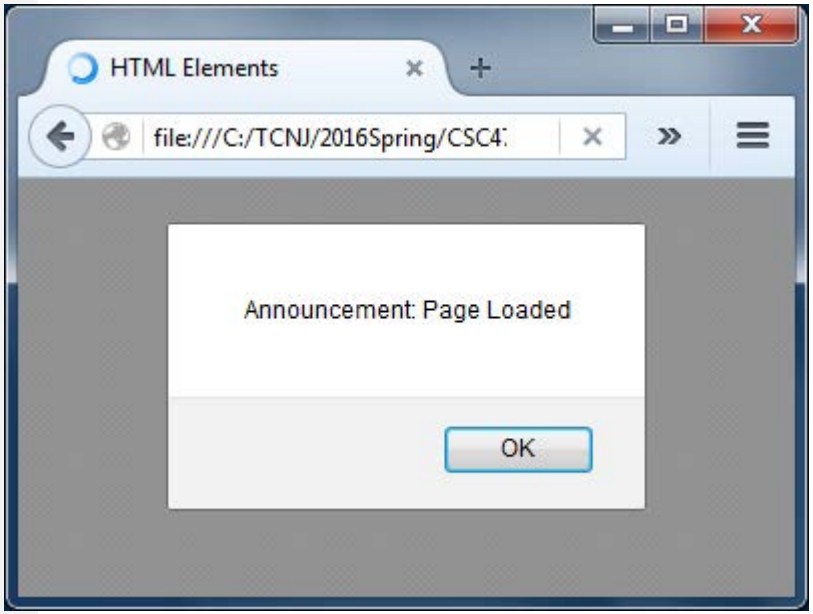
# Some Common HTML Tags: JavaScript

- JavaScript may be associated with a web page using the `<script>` tag
- `<script>` tags must be closed with `</script>`
- Arbitrary JavaScript source code may appear between tags
- Alternatively, may reference an external JavaScript file which will be loaded into browser
- Scripts run as soon as tag is encountered
  - `<script>` tags often are placed at end of `<body>` when DOM objects are accessed by script
  - Putting `<script>` at end ensures that DOM objects are loaded
  - One page may have multiple `<script>` tags
- Browsers limit what JavaScript can access – for security reasons
  - JavaScript in a browser runs in sandbox
  - Ex. No access to clipboard

04/html_tags.html

# Some Common HTML Tags: JavaScript

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>HTML Elements</title>

    <script type="text/javascript">
      function announce(msg) {
        alert("Announcement: " + msg);
      }
    </script>

  </head>
  <body>

    <script type="text/javascript">
      announce("Page Loaded");
    </script>

  </body>
</html>
```




04/script.html

# Cascading Style Sheets

- Cascading Stylesheets (CSS) is a language for styling HTML and other markup content
- The primary goal of CSS is to allow separation of a document's presentation characteristics (formatting) from the document's content.

<u>Example</u>

- Content -> A paragraph of text
- Style -> font size/color/style/weight, text alignment,
  background color, border color/size, margins, padding, …

- Styles are specified in "style sheets" which may be (1) included in an HTML page or (2) stored in a separate document and loaded by the document
- Styles may be applied to content using (1) CSS rules or (2) the "style" attribute of an HTML tag
- CSS is the preferred way to style page content instead of "presentation HTML"

https://docs.webplatform.org/wiki/css/tutorials

# Style Sheets

- Whether in an HTML document or as a separate document, Style Sheets are composed of CSS Rules
- CSS Rules relate HTML tags, attributes and structure to the styles that should be applied during page rendering

```
selector {
  property1 : value1;
  property2 : value2;
  property3 : value3;
}
```

- Example CSS Rule – all <p> tags should have a margin of 5 pixels, and use an Arial font with blue text

```
/* apply style to all <p> tags */
p {
  margin      : 5px;
  font-family : arial;
  color       : blue;
}
```

CSS Comment

https://docs.webplatform.org/wiki/guides/getting_started_with_css

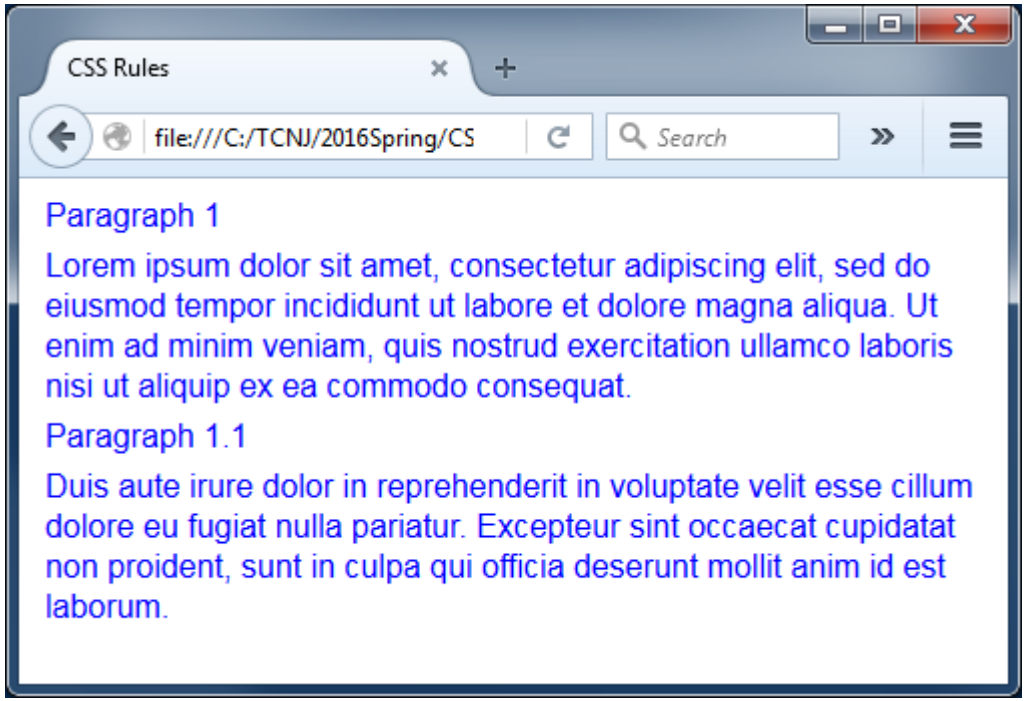
# Applying Styles to Tags

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>CSS Rules</title>

    <style>
      /* apply style to all <p> tags */
      p {
        margin      : 5px;
        font-family : arial;
        color       : blue;
      }
    </style>

  </head>
  <body>
    <p>Paragraph 1</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
    </p>

    <p id='para11'>Paragraph 1.1</p>
    <p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
    eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
    sunt in culpa qui officia deserunt mollit anim id est laborum.
    </p>

  </body>
</html>
```

Embedded style sheet in `<style>` tags

04/css_rules1.html

# Applying Styles to Tags




*Rendered Page*

04/css_rules1.html

# Applying Styles to Tags by Class

- A CSS selector can refer to all tags with a specific `class` attribute value
- Precede the `class` attribute value with a "`.`"
- Useful when wanting to specify a shared style for a range of tags
- You are free to define class names.

```
.class {
  property1 : value;
  property2 : value;
  property3 : value;
}
```

```
/* apply style to all elements with
   class='highlight' attribute */

.highlight {
    background-color : yellow;
}
```

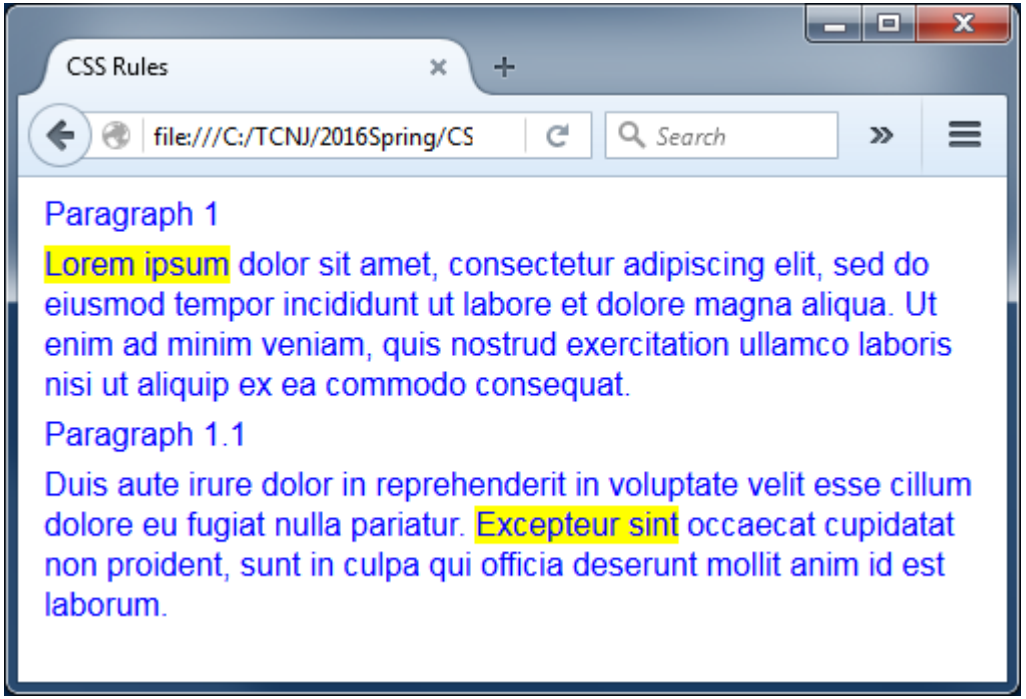
# Applying Styles to Tags by Class

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>CSS Rules</title>

    <!-- Embedded Style Sheet -->
    <style>
      /* apply style to all <p> tags */
      p {
        margin      : 5px;
        font-family : arial;
        color       : blue;
      }

      /* apply style to all elements with class='highlight' attribute */
      .highlight {
        background-color : yellow;
      }

    </style>
  </head>
  <body>
    <p>Paragraph 1</p>
    <p><span class='highlight'>Lorem ipsum</span> dolor sit amet, consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
    </p>

    <p>Paragraph 1.1</p>
    <p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat
    nulla pariatur. <span class='highlight'>Excepteur sint</span> occaecat cupidatat non proident,
    sunt in culpa qui officia deserunt mollit anim id est laborum.
    </p>

  </body>
</html>
```

04/css_rules2.html

# Applying Styles to Tags by Class

*Rendered Page*




04/css_rules2.html

# Applying Styles to Tag by Attribute ID

- A CSS selector can refer to a single tag with a specific `id` attribute value
- Precede the `id` attribute value with a "`#`"
- Useful when wanting to specify a style of a specific tag
- You are free to define id's. Make each unique to a document.

```
#id {
  property1 : value;
  property2 : value;
  property3 : value;
}
```

```
/* apply style to element with id='para1' attribute */
#para1 {
    font-size : 14pt;  /* overrides p */
    color     : red;
}
```

# CSS Colors

CSS properties such as `color` and `background-color` may be assigned to a color vale

There are multiple ways to specify a CSS color value:

1. name
   - Predefined color names are defined
   - `color: red; background-color: white;`

2. RGB values may be specified using hexadecimal values (00 to FF)
   - Begin hex color values with a '#'
   - `color: #FF0000; background-color: #FFFFFF;`

3. `rgb(R, G, B)` and `rgba(R, G, B, A)` functions
   - `R, G, B` can be values in `[0, 255]`
   - `A` is opacity, a value in `[0.0, 1.0]`

# CSS Color Names

| Color | Keyword | RGB hex values |
|---|---|---|
| | black | #000000 |
| | silver | #c0c0c0 |
| | gray | #808080 |
| | white | #ffffff |
| | maroon | #800000 |
| | red | #ff0000 |
| | purple | #800080 |
| | fuchsia | #ff00ff |
| | green | #008000 |
| | lime | #00ff00 |
| | olive | #808000 |
| | yellow | #ffff00 |
| | navy | #000080 |
| | blue | #0000ff |
| | teal | #008080 |

...

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

# Color String Options

```
Color name (red, green, cornflowerblue, etc.)
#••• — shortened HTML color: (#000, #fc0, etc.)
#•••••• — full length HTML color: (#000000, #bd2300)
rgb(•••, •••, •••) — red, green and blue channels values: (rgb(200, 100, 0))
rgba(•••, •••, •••, •••) — also with opacity
rgb(•••%, •••%, •••%) — same as above, but in %: (rgb(100%, 175%, 0%))
rgba(•••%, •••%, •••%, •••%) — also with opacity
hsb(•••, •••, •••) — hue, saturation and brightness values: (hsb(0.5, 0.25, 1))
hsba(•••, •••, •••, •••) — also with opacity
hsb(•••%, •••%, •••%) — same as above, but in %
hsba(•••%, •••%, •••%, •••%) — also with opacity
hsl(•••, •••, •••) — hue, saturation and luminosity values: (hsb(0.5, 0.25,
0.5))
hsla(•••, •••, •••, •••) — also with opacity
hsl(•••%, •••%, •••%) — same as above, but in %
hsla(•••%, •••%, •••%, •••%) — also with opacity
```

Note that % can be used any time: `rgb(20%, 255, 50%)`.

http://snapsvg.io/docs/#Snap.getRGB

# Applying Styles to Tags by ID

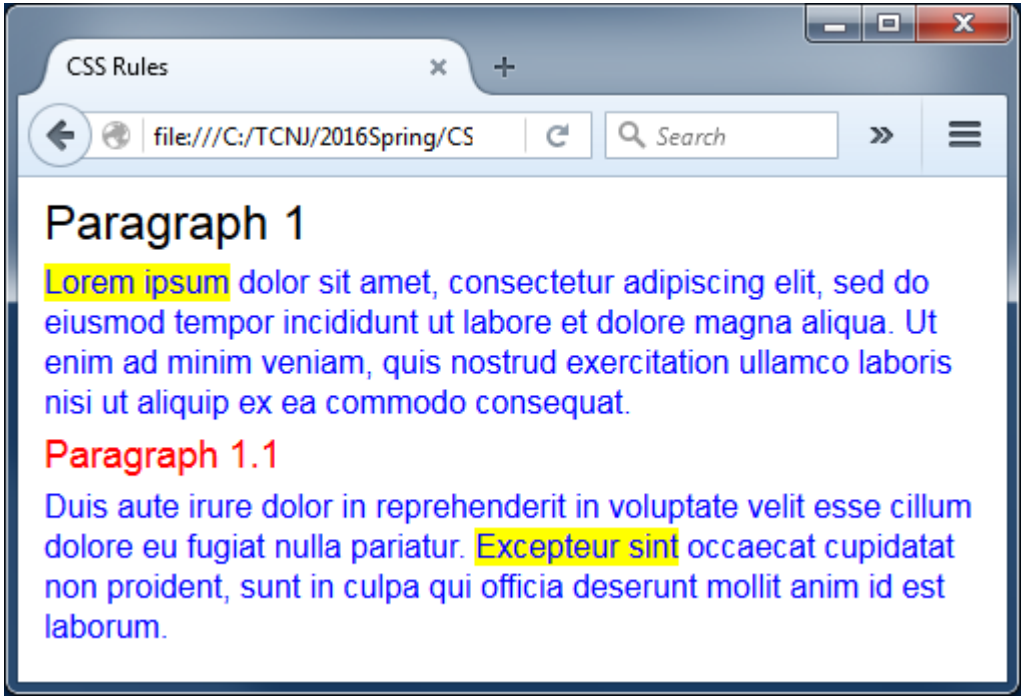
```
<meta charset="UTF-8">
<title>CSS Rules</title>

<!-- Embedded Style Sheet -->
<style>
  /* apply style to all <p> tags */
  p {
    margin      : 5px;
    font-family : arial;
    color       : blue;
  }

  /* apply style to all elements with class='highlight' attribute */
  .highlight {
    background-color : yellow;
  }

  /* apply style to element with id='para1' attribute */
  #para1 {
    font-size : 18pt;   /* styles override <p> */
    color     : black;
  }

  /* apply style to element with id='para11' attribute */
  #para11 {
    font-size : 14pt;   /* styles override <p> */
    color     : red;
  }

</style>
</head>
<body>
  <!-- This file demonstrates various scenarios for the application of CSS -->

  <p id='para1'>Paragraph 1</p>
  <p><span class='highlight'>Lorem ipsum</span> dolor sit amet, consectetur adipiscing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
  quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
  </p>

  <p id='para11'>Paragraph 1.1</p>
  <p>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat
  nulla pariatur. <span class='highlight'>Excepteur sint</span> occaecat cupidatat non proident,
  sunt in culpa qui officia deserunt mollit anim id est laborum.
  </p>
```

Styles cascade with well-defined rules – more general to more specific or more important

04/css_rules3.html

# Applying Styles to Tags by ID

*Rendered Page*




04/css_rules3.html

# Cascading Styles

CSS Rules Window

Inspector - file:///C:/TCNJ/2016Spring/CSC470/Lectures/04/css_rules.html

Inspe... Con... Debu... Style Ed... Performa... Net...

html > body > p#para1

Rules | Computed | Fonts

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <!--This file demonstrates various scenarios for the …-->
    <p id="para1">Paragraph 1</p>
    <p></p>
    <p id="para11">Paragraph 1.1</p>
    <p></p>
  </body>
</html>
```

Filter Styles

```
element {                    inline
}
#para1 {                     inline:22
    font-size: 18pt;
    color: #000;
}
p {                          inline:10
    margin: 5px;
    font-family: arial;
    ~~color: #00F;~~
}
```

Net | CSS | JS | Security | Logging | Server | Clear | Filter output

`color` style defined in `p` rule is overridden by `color` defined in the `#para1` rule

04/css_rules3.html

# Linking to External Style Sheets

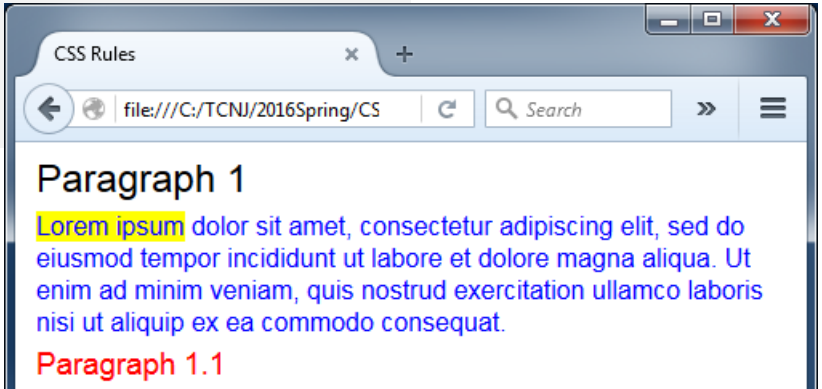
- Move all styles to a separate file, without <style> tags
- Add a `<link>` tag to `<head>` indicating where the stylesheet is located

```
<link href="path/to/styles.css" rel="stylesheet">
```

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>CSS Rules</title>

    <!-- Linked Style Sheet -->
    <link href="css_rules.css" rel="stylesheet">

  </head>
  <body>
…
```

# Linking to External Style Sheets

```
/* apply style to all <p> tags */
p {
  margin      : 5px;
  font-family : arial;
  color       : blue;
}

/* apply style to all elements with class='highlight' attribute */
.highlight {
  background-color : yellow;
}

/* apply style to element with id='para1' attribute */
#para1 {
  font-size : 18pt;   /* styles override <p> */
  color     : black;
}

/* apply style to element with id='para11' attribute */
#para11 {
  font-size : 14pt;   /* styles override <p> */
  color     : red;
}
```




04/css_rules.css

# Inline Styles, the `style` Attribute

- Styles may be applied directly to a tag without rules
- Add a `style` attribute to a tag and set its value to the styles that should be applied to the tag

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>CSS Rules</title>
  </head>
  <body>
    <p style="font-family: arial; font-size: 18pt; margin: 5px;">Paragraph 1</p>
    <p style="font-family: arial; color: blue; margin: 5px;"><span style="background-color: yellow">Lorem ipsum</spar
    dolor sit amet, consectetur adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
    quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
    </p>

    <p style="font-family: arial; color: red; font-size: 14pt; margin: 5px;">Paragraph 1.1</p>
    <p style="font-family: arial; color: blue; margin: 5px;">Duis aute irure dolor in reprehenderit in voluptate
    velit esse cillum dolore eu fugiat
    nulla pariatur. <span style="background-color: yellow">Excepteur sint</span> occaecat cupidatat non proident,
    sunt in culpa qui officia deserunt mollit anim id est laborum.
    </p>

  </body>
</html>
```

04/css_rules4.html

# CSS Selectors

Additional CSS selector options exist for searching/selecting DOM tree Elements

**Type Selectors**

Matches Elements by tag name
`h1` matches all `<h1></h1>` in the tree

**Class Selectors**

Matches Elements with a matching class attribute.
Selector begins with a "`.`"
Used to find all Elements of a class
`.title` matches all `<tag class="title">`*`</tag>`* in tree

**ID Selectors**

Matches Elements with a matching id attribute
Selector begins with a "`#`"
Used to find individual Elements in the DOM
`#elem1` matches the `<tag id="elem1">`*`</tag>`* in tree

**Universal Selector**

Matches any element in tree
Selector is "`*`"
`*` applies to all elements

https://www.w3.org/TR/REC-CSS2/selector.html

# CSS Selectors

**Descendant Selectors**

Matches Elements that are descendants of another element in the tree

Made up of selectors separated by a space

`p li` matches all `<li></li>` Elements under a `<p></p>`

**Child Selectors**

Matches an Element that is a direct child of another Element

Made up of selectors separated by a ">"

`ol > li` matches all `<li></li>` Elements directly under an `<ol></ol>`

**Sibling selectors**

Matches the sibling Element immediately following an Element

Made up of selectors separated by a "+"

`h2 + h3` matches all `<h3></h3>` that immediately follow an `<h2></h2>`

https://www.w3.org/TR/REC-CSS2/selector.html

# CSS Selectors

## Attribute Selectors

Matches Elements based on their attributes or attribute values

Follow selector with `[…]` containing an attribute name or attribute="value"

`img[title]` matches all `<img title="…"></img>` Elements

`img[src="bozo.png"]` matches all `<img src="bozo.png">` Elements

## Pseudo-classes

Matches Elements based on their state (not any value)

Selector is made up of a selector with "`:`" followed by state

E.g.

- `a:link` is the selector for normal `<a></a>` Elements
- `a:visited` is the selector for visited `<a></a>` Elements
- `a:hover` is the selector for hover state of `<a></a>` Elements
- `a:active` is the selector for active `<a></a>` Elements
- `a:focus` is the selector value for the `<a></a>` Element currently with focus

Other CSS selector options are available

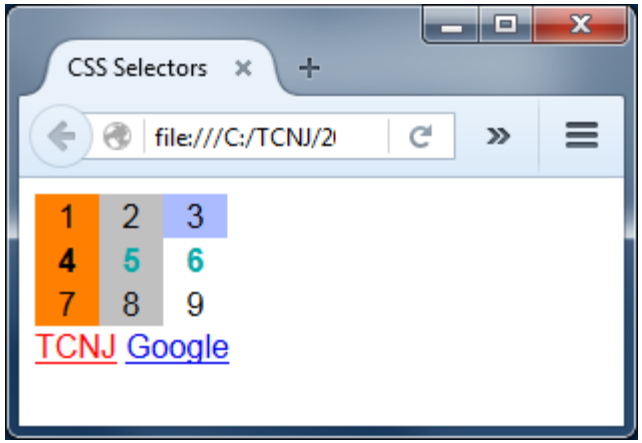
https://www.w3.org/TR/REC-CSS2/selector.html

# CSS Selector Example

```html
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS Selectors</title>
    <link href="selectors.css" rel="stylesheet">
  </head>
  <body>
    <table>
      <tbody>
        <tr id="row1">
          <td class="col1">1</td>
          <td>2</td>
          <td id="td13">3</td></tr>
        <tr id="row2">
          <td class="col1">4</td>
          <td> <span>5<span> </td>
          <td> <span>6<span> </td>
        </tr>
        <tr id="row3">
          <td class="col1">7</td>
          <td>8</td>
          <td>9</td>
        </tr>
      </tbody>
    </table>
    <div>
      <a href="http://tcnj.edu">TCNJ</a>
      <a href="http://www.google.com">Google</a>
    </div>
  </body>
</html>
```

```css
/* all elements */
* {
  font-family: arial;
}
/* all table elements */
table {
  border-collapse: collapse;
  font-size: 12pt;
}
/* all td elements */
td {
  width:30px;
  text-align: center;
}
/* with id='td13' */
#td13 {
  background-color: #AABBFF;
}
/* with class='col1' */
.col1 {
  background-color: rgb(255, 127, 0);
}
/* td direct child of id='row2' */
#row2 > td {
  font-weight: bold;
}
/* all span children of id='row2' */
#row2 span {
  color: #0AA;
}
/* immediate td sibling of class='col1' */
.col1 + td {
  background-color: #C0C0C0;
}
/* a elements hovered over */
a:hover {
  font-size: 24pt;
}
/* a elements with href="http://tcnj.edu" */
a[href="http://tcnj.edu"] {
  color: red;
}
```

# CSS Selector Example



# DOM Classes (Interfaces)

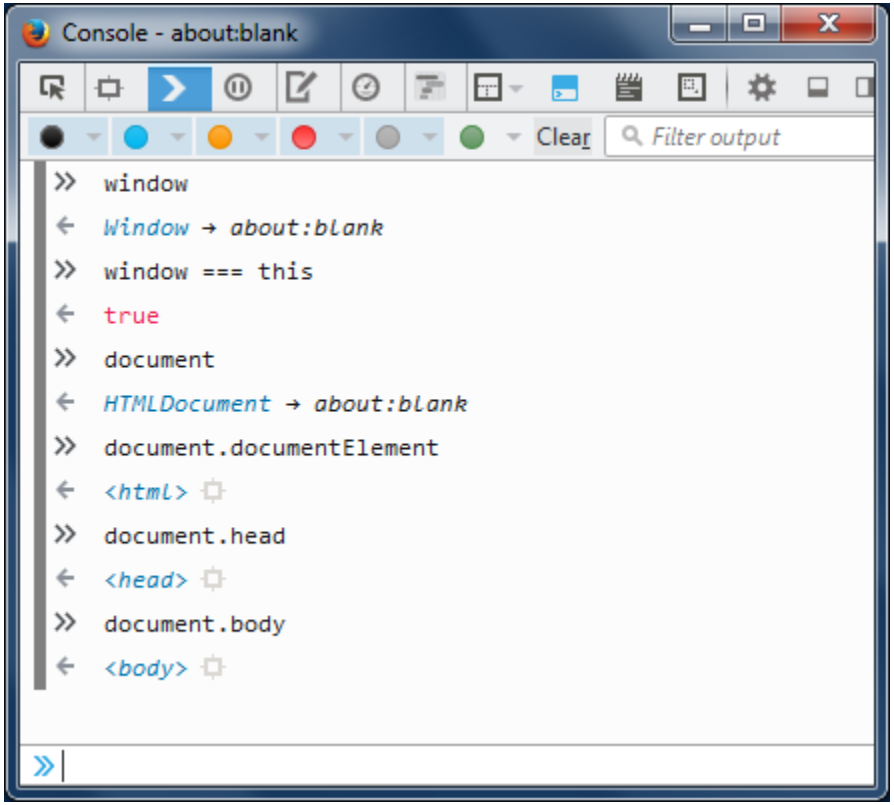
- Node
  - an interface from which many DOM types inherit
- Document
  - represents a web page loaded in the browser
  - serves as an entry point into the web page's content, the DOM tree
  - inherits Node
- Element
  - represents an object in a Document
  - describes methods and properties common to all kinds of elements
  - inherits Node
- Window
  - represents a window containing a DOM document
  - `window.document` points to current Document
  - The JavaScript global object in a Browser
  - `this === window  // -> true in global scope`

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

# JavaScript in the Browser

JavaScript in the Browser begins with several predefined globals

- JavaScript's global object in the browser is `window`
- `document` (a `window` attribute) references the DOM top-level node
- `document.documentElement` refers to the object representing the `<html>` tag
- `document.head` refers to `<head>`
- `document.body` refers to `<body>`



# Traversing the DOM w/ JavaScript

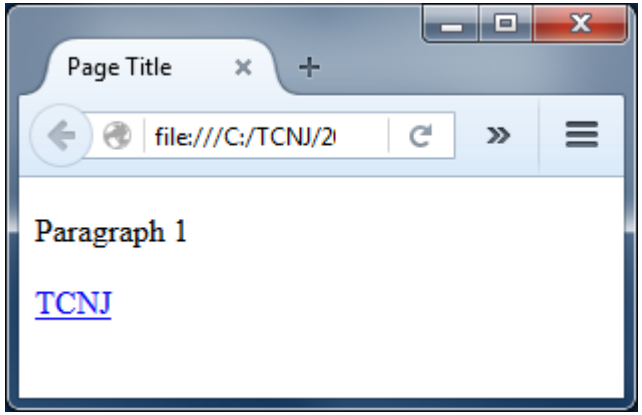
- The browser translates HTML tags into objects and stores as a tree
- Element and Node objects provide properties and methods that support DOM navigation

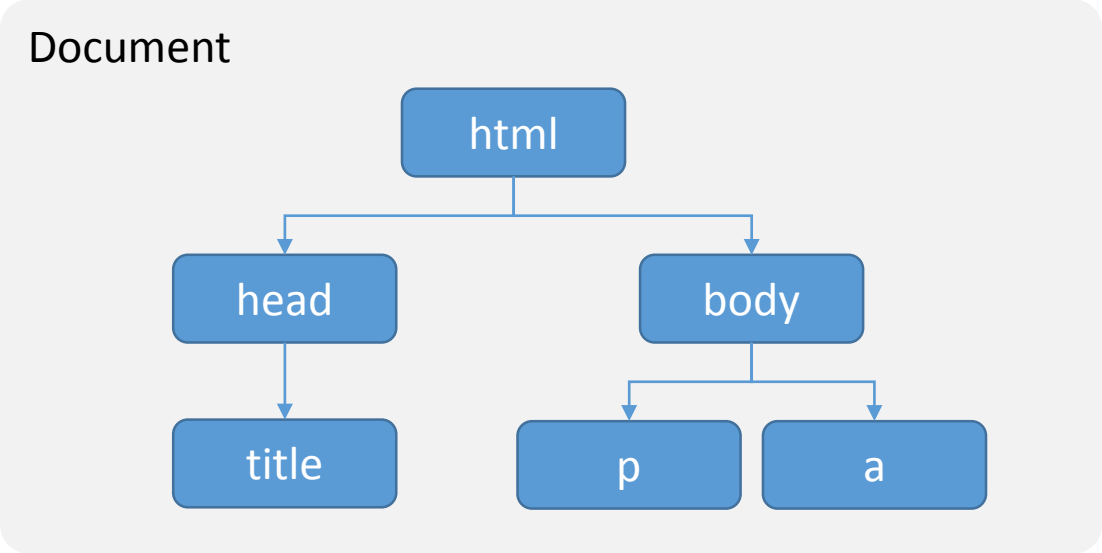
  - `anElement.children`
    - Returns a live HTMLCollection of all Elements only.
    - Array-like object supporting `[]` element access and `length` property
  - `aNode.childNodes`
    - Returns a live NodeList collection of all the child Nodes of this node.
    - Array-like object supporting `[]` element access and `length` property
  - `aNode.parentNode`
    - Returns a Node that is the parent of this node, or `null`.
  - `aNode.firstChild`, `aNode.lastChild`
    - Returns a Node representing the first/last direct child node of the node, or `null`
  - `aNode.previousSibling`, `aNode.nextSibling`
    - Returns a Node representing the next node in the tree, or `null`
  - `aNode.hasChildNodes()`
    - Returns a Boolean indicating if the element has any child nodes, or not.

https://developer.mozilla.org/en-US/docs/Web/API/Node, https://developer.mozilla.org/en-US/docs/Web/API/Element

# Traversing the DOM w/ JavaScript

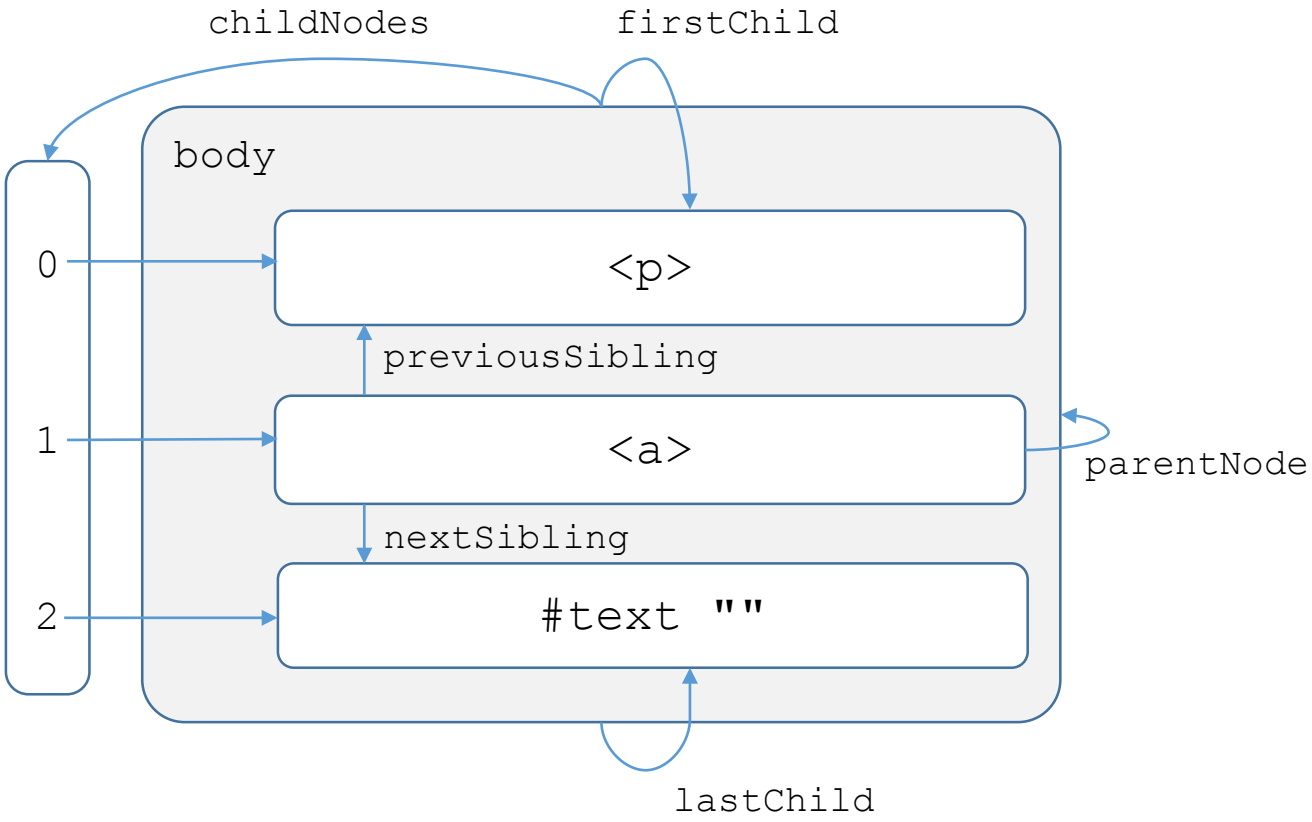
```
<!doctype html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body><p>Paragraph 1</p><a href="http://tcnj.edu">TCNJ</a></body>
</html>
```




*Removing newlines and spaces from the HTML eliminates Text Nodes from the DOM

# Traversing the DOM w/ JavaScript




http://eloquentjavascript.net/13_dom.html

04/sample.html

# Node Type Constants

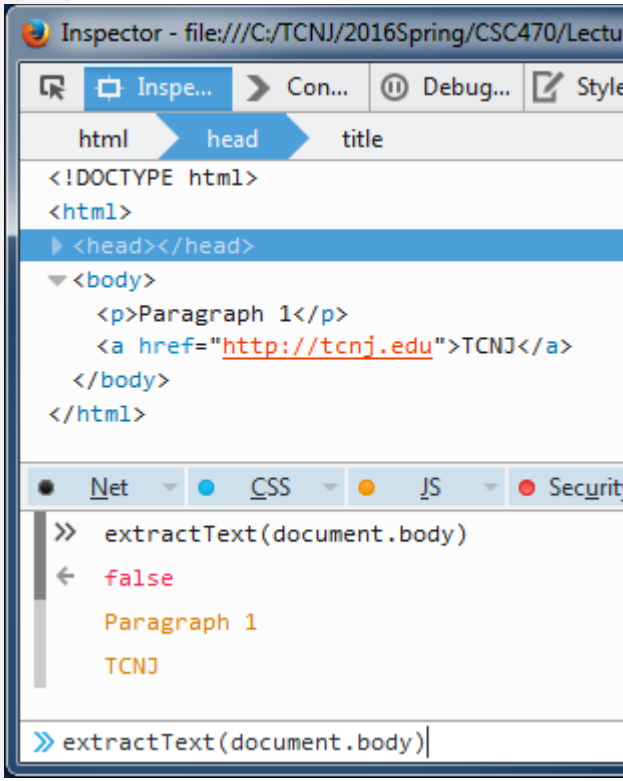
- Node objects possess properties that indicate their name, type, and value
  - `aNode.nodeValue`
  - `aNode.nodeName`
  - `aNode.nodeType`

- Types may be detected using constants defined in the Node object
  - `Node.ELEMENT_NODE` // 1, an HTML tag
  - `Node.TEXT_NODE` // 3, any text or whitespace
  - `Node.COMMENT_NODE` // 8, <!-- -->
  - `Node.DOCUMENT_NODE` // 9, The main Document node
  - `Node.DOCUMENT_TYPE_NODE` // 10, <!doctype html>
  - …

# Recursive Tree Traversal

Extracting all text nodes from a page

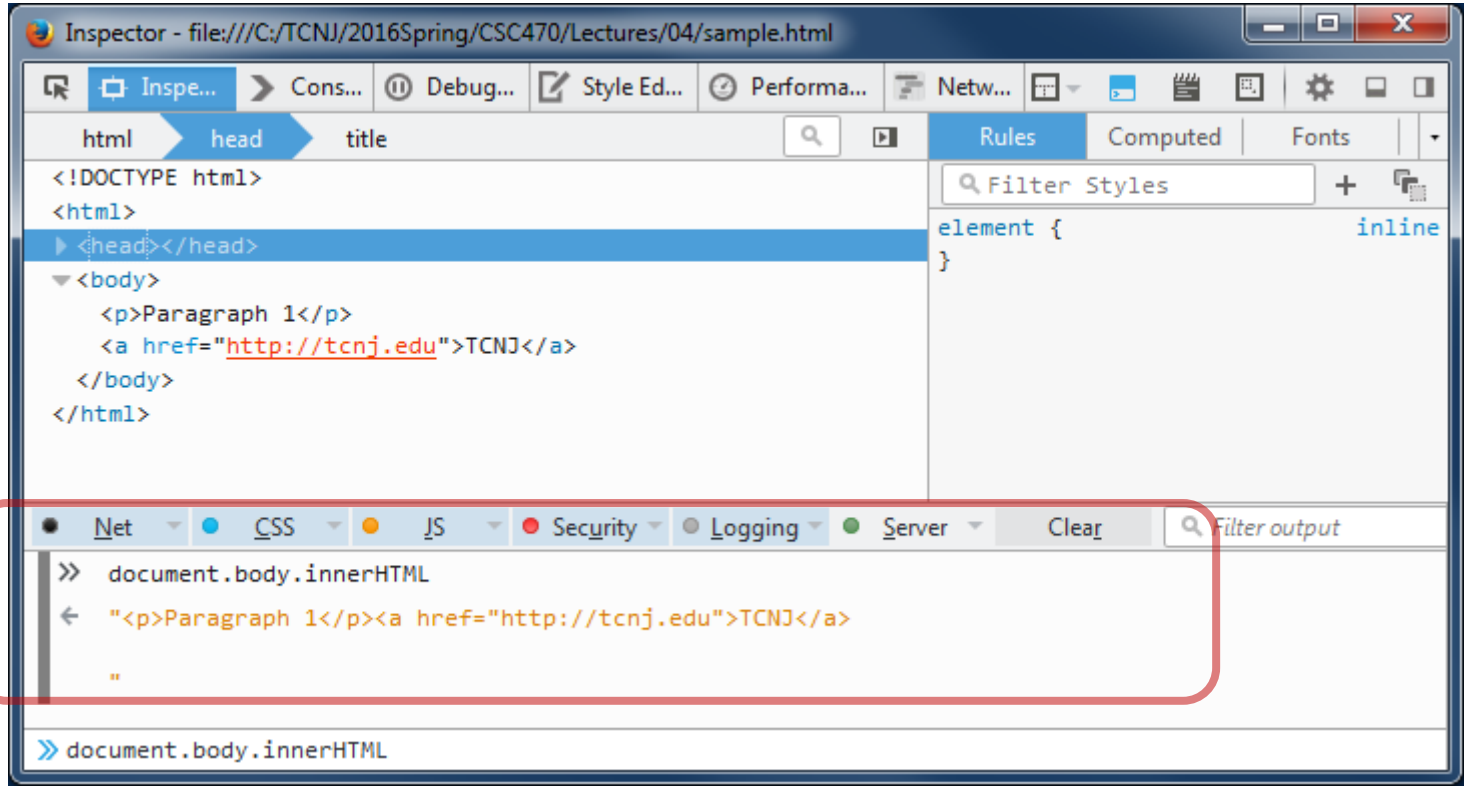
```
// extractText.js

function extractText(node) {
  if (node.nodeType === document.ELEMENT_NODE) {
    for (var i = 0; i < node.childNodes.length; i++) {
      extractText( node.childNodes[i] );
    }
    return false;
  } else if ( node.nodeType === document.TEXT_NODE ) {
    var txt = node.nodeValue.trim();
    if (txt.length > 0) {
      console.log( node.nodeValue );
    }
  }
}

extractText(document.body);
```




04/extractText.js

# Element `innerHTML` Property

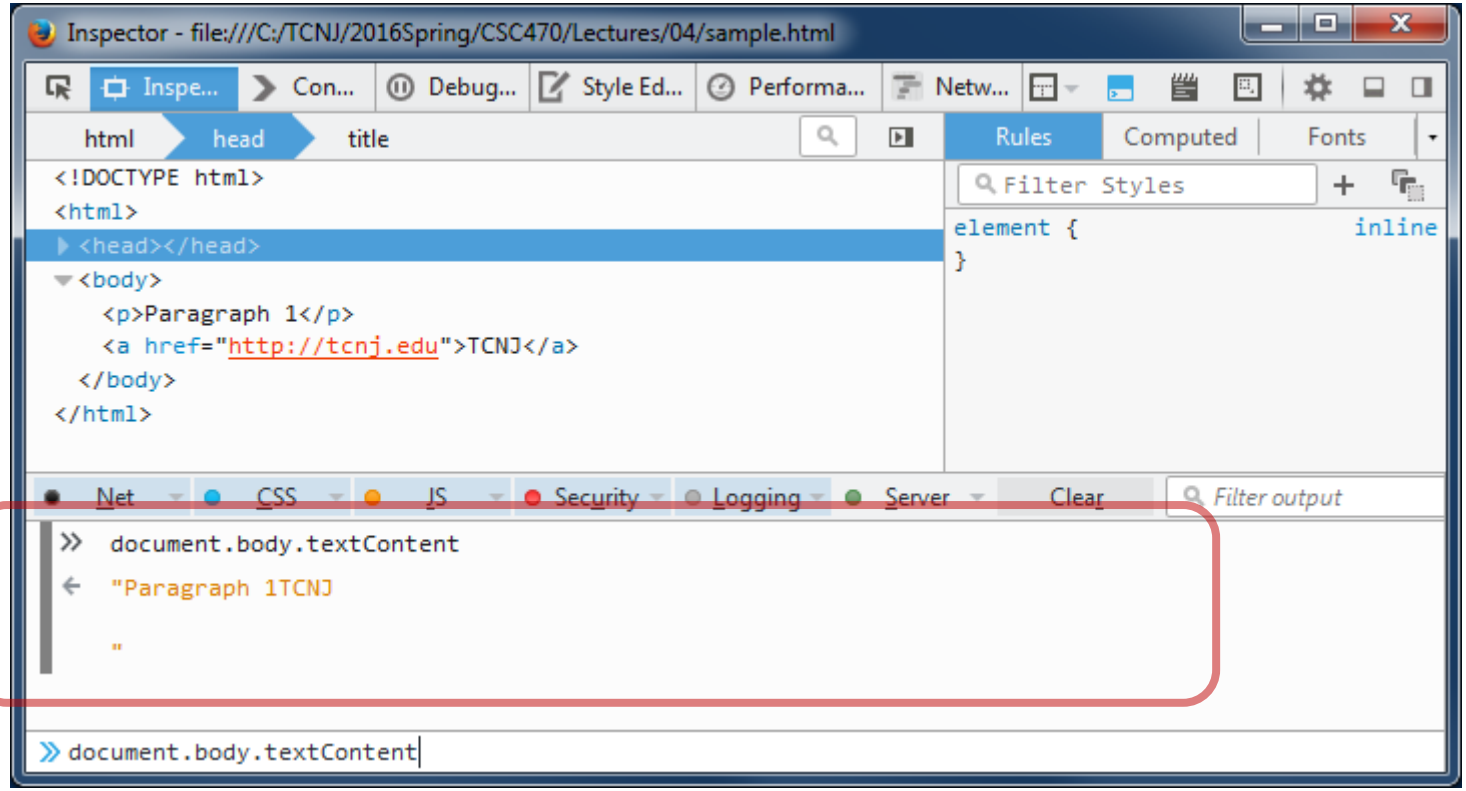
- An Element object's `innerHTML` property represents the HTML string of the element's descendants.




04/sample.html

# Node `textContent` Property

- A Node object's `textContent` property represents the text content of the node and its descendants.




04/sample.html

# Selecting DOM Elements

Collect all elements with given tag name

- `aDocOrElement.getElementsByTagName("tag")`

Collect elements with given class attribute value

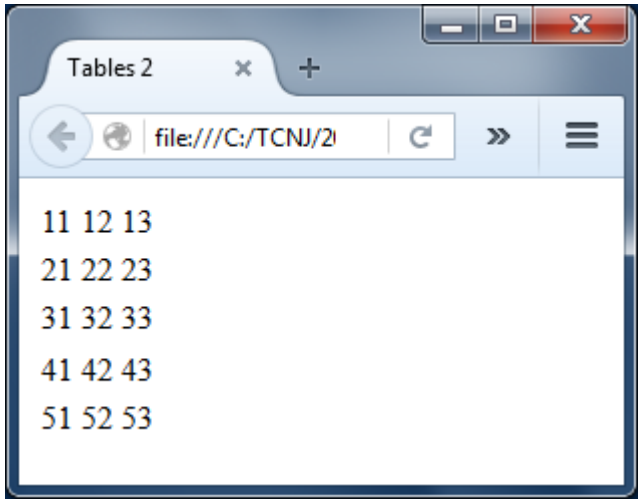
- `aDocOrElement.getElementsByClassName("class")`

Get a single element given an id attribute value

- `aDocOrElement.getElementById("id")`

# Selecting DOM Elements

```
<table id="tab1">
  <tbody>
    <tr class="odd">
      <td>11</td><td>12</td><td>13</td>
    </tr>
    <tr class="even">
      <td>21</td><td>22</td><td>23</td>
    </tr>
    <tr class="odd">
      <td>31</td><td>32</td><td>33</td>
    </tr>
  </tbody>
</table>

<table id="tab2">
  <tbody>
    <tr class="odd">
      <td>41</td><td>42</td><td>43</td>
    </tr>
    <tr class="even">
      <td>51</td><td>52</td><td>53</td>
    </tr>
  </tbody>
</table>
```




- Two tables with id attributes
- Rows with odd/even class

04/table2.html

# Selecting DOM Elements

C:\TCNJ\2016Spring\CSC470\Lectures\04\table2.js

File Edit View Execute Environment Help

Open File... Save Save As... Run Inspect

```
1  // table2.js
2
3  // Get all tr elements
4  var getAllRows = function() {
5    var rows = document.getElementsByTagName("tr");
6    return rows;
7  };
8
9  // Get rows by class cls
10 var getRows = function(cls) {
11   var rows = document.getElementsByClassName(cls);
12   return rows;
13 };
14
15 // Get the table with the given id
16 var getTable = function(id) {
17   var tab = document.getElementById(id);
18   return tab;
19 };
20
21 // Test
22 var r1 = getAllRows();
23 console.log(r1);
24
25 var r2 = getRows("odd");
26 console.log(r2);
27
28 var t1 = getTable("tab2");
29 console.log(t1);
30
```

Line 30, Col 1

Console - file:///C:/TCNJ/2016S...

Clear

```
HTMLCollection [    Scratchpad/1:21:1
<tr.odd> ,
<tr.even> ,
<tr.odd> ,
<tr.odd> ,
<tr.even> ]
HTMLCollection [    Scratchpad/1:24:1
<tr.odd> ,
<tr.odd> ,
<tr.odd> ]
<table              Scratchpad/1:27:1
id="tab2"> 
```

04/table2.js

# Selectors

- In addition to tag name, class and id, elements may be selected out of the DOM using more complex conditions, known as CSS selectors

| Selector | Selects |
|---|---|
| `TAG` | Any E element with tag name TAG |
| `.CLASS` | Any E element with class attribute value CLASS |
| `#ID` | E element with id attribute value ID |
| `A E` | Any E element that is a descendant of an A element (that is: a child, or a child of a child, etc.) |
| `A > E` | Any E element that is a child (i.e. direct descendant) of an A element |
| `E:first-child` | Any E element that is the first child of its parent |
| `B + E` | Any E element that is the next sibling of a B element (that is: the next child of the same parent) |

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_Started/Selectors

# Selecting DOM Elements

Returns the first Element node within the document, in document order, that matches the specified selectors.

- `aDocOrElement.querySelector("selector")`

Returns a list of all the Element nodes within the document that match the specified selectors.

- `aDocOrElement.querySelectorAll("selector")`

https://developer.mozilla.org/en-US/docs/Web/API/Document

# Selecting DOM Elements

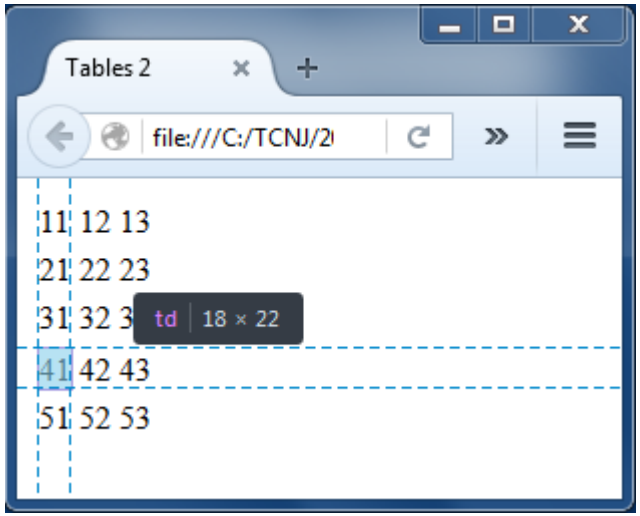
C:\TCNJ\2016Spring\CSC470\Lectures\04\table3.js

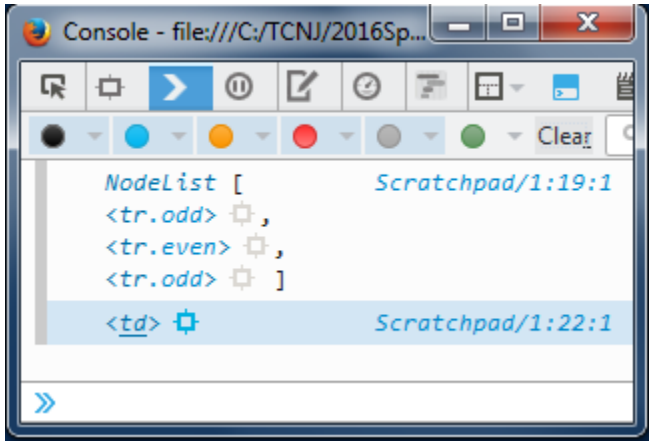
File Edit View Execute Environment Help

Open File... Save Save As... Run Insp

```
// table3.js

// Get all tr elements in table with id
var getTableRows = function(id) {
  var sel = '#' + id + " tr";
  var rows = document.querySelectorAll(sel);
  return rows;
};

// Get first td element in table with id
var getTableTd = function(id) {
  var sel = '#' + id + " td";
  var td = document.querySelector(sel);
  return td;
};

// Test
var r3 = getTableRows('tab1');
console.log(r3);

var td1 = getTableTd("tab2");
console.log(td1);
```

Line 23, Col 1




04/table3.js

# Modifying the DOM

Nearly everything about the DOM data structure can be changed.

- Nodes may be…
  - created
  - added
  - deleted
  - cloned from another node
  - inserted before another node
  - appended after a node
  - used to replace an existing node
  - deleted
  - added as a new child to a node
- Node attributes may be…
  - read
  - modified
  - added
  - removed

# Modifying the DOM: Creating Elements

Create a new Element given a string that specifies the type of element to be created

- `document.createElement(tagName)`

Create a new Text node

- `document.createTextNode(text)`

Create a new comment node, and returns it.

- `document.createComment(text)`

Clone a Node, and optionally, the children of the node. By default, deep = true

- `aNode.cloneNode(deep)`

- Set (or get) part of the DOM tree as HTML text
  - `anElement.innerHTML = html`

https://developer.mozilla.org/en-US/docs/Web/API/Document

# Modifying the DOM: Rearranging the Tree

Adds a node to the end of the list of children of a specified parent node

- `aParentNode.appendChild(aChild)`

Inserts specified node before the reference node as a child of the current node

- `aParentNode.insertBefore(newNode, referenceNode)`

Replaces one child node of the specified element with another. Returns replaced node.

- `aNode.replaceChild(newChild, oldChild)`

Removes a child node from the DOM. Returns removed node.

- `aNode.removeChild(child)`

Returns a Boolean value indicating whether the current Node has child nodes or not.

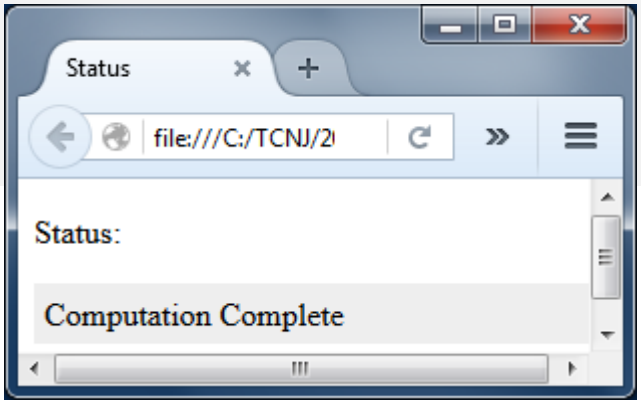
- `aNode.hasChildNodes()`

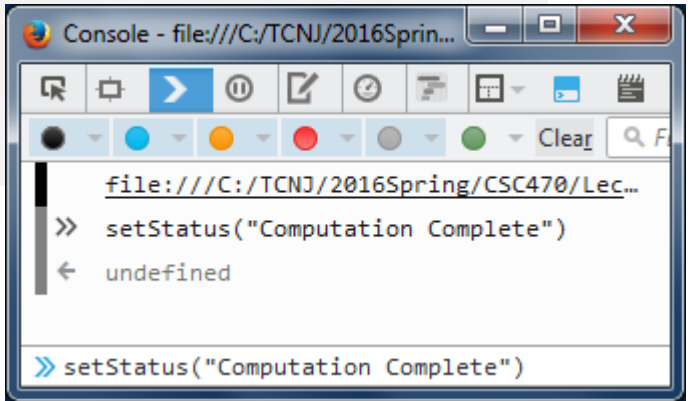
https://developer.mozilla.org/en-US/docs/Web/API/Element

# Example: A status message

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Status</title>
    <style>
      #status {
        width: 100%;
        padding: 5px;
        background-color: #eee;
      }
    </style>
    <script type="text/javascript" src="status.js"></script>
  </head>
  <body>

  <p>Status:</p>
  <p id="status"> </p>

  </body>
</html>
```

```javascript
// status.js

// Change the status message
var setStatus = function(msg) {
  var el = document.getElementById('status');
  el.innerHTML = msg;
};
```



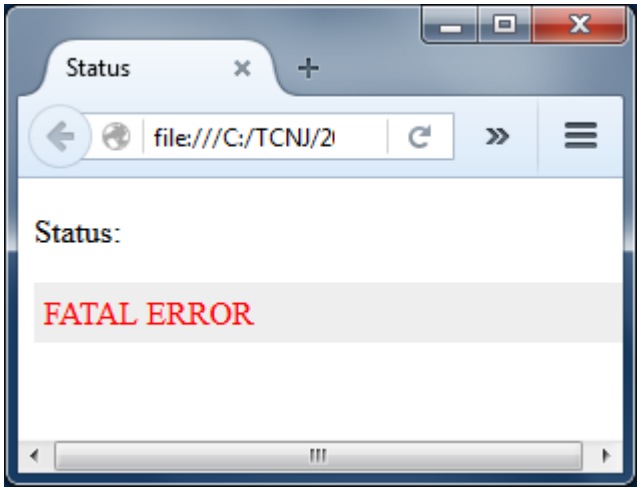

04/status.html

# Example: A formatted status message

- The `innerHTML` property may be assigned to any HTML string.
- The string will be parsed into Nodes and appended to the element.

```
// status.js

// Change the status message
var setStatus = function(msg) {
  var el = document.getElementById('status');
  el.innerHTML = msg;
};
```

```
setStatus("<span style='color:red;'>FATAL ERROR</span>")
```




04/status.html

# Example: Logging

Log

file:///C:/TCNJ/2016Sprin

Search

Log

Data processing initiated

Phase 1 complete

Phase processing completed

CSC470/Lectures/04/log.html

Debugger | Style Editor | Performance | Network

Search scripts (Ctrl+P)

```
// log.js

var addLog = function(msg) {
  // Add a new <p> with a text node to the end of <body>

  // Create new nodes
  var p = document.createElement('p');
  var txt = document.createTextNode(msg);

  // Add text node to p
  p.appendChild(txt);

  // Add p to body
  document.body.appendChild(p);
};
```

Net | CSS | JS | Security | Logging | Server | Clear | Filter

```
addLog("Phase processing completed")
undefined
```

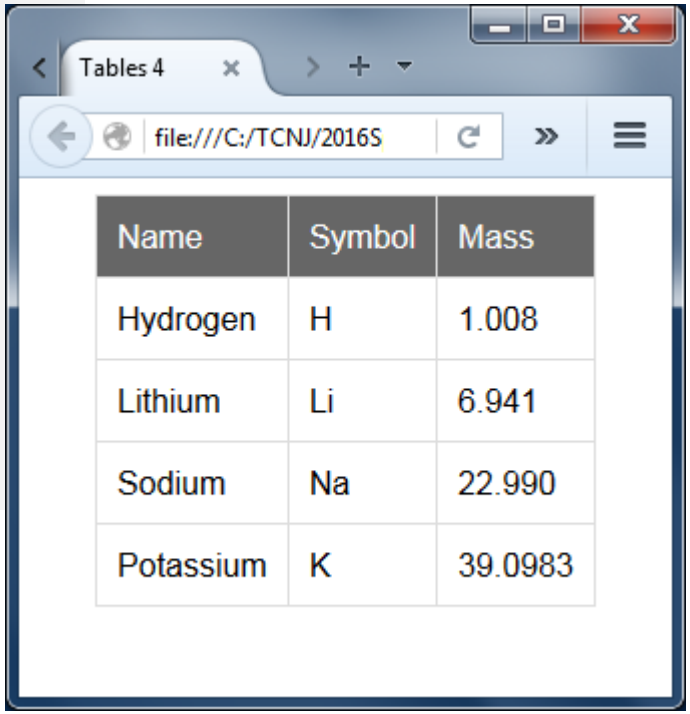
```
addLog("Phase processing completed")
```

04/log.html

# Example: Clone a Table Row

```
// table4.js

// Add a row to the elements table
var addRow = function(name, symbol, mass) {

  // Get the first element in a list of tbody elements
  var tbody = document.getElementsByTagName('tbody')[0];

  // Get number of tr elements
  var ntr = tbody.children.length;

  // Deep clone the first child
  var trClone = tbody.children[ntr-1].cloneNode(true);

  // Modify contents of cloned tr
  trClone.children[0].innerHTML = name;
  trClone.children[1].innerHTML = symbol;
  trClone.children[2].innerHTML = mass;

  // Append to the end
  tbody.appendChild(trClone);
};
```

```
addRow("Potassium", "K", "39.0983")
```




| Name | Symbol | Mass |
|---|---|---|
| Hydrogen | H | 1.008 |
| Lithium | Li | 6.941 |
| Sodium | Na | 22.990 |
| Potassium | K | 39.0983 |

04/table4.html

# Element Attribute Manipulation

Returns a live `NamedNameMap` collection of all attribute nodes registered to the specified node.

- `anElement.attributes`

Returns a Boolean value, `true` or `false`, indicating if the current element has any attributes or not

- `anElement.hasAttributes()`

Returns a Boolean value indicating whether the specified element has the specified attribute named `attName`, or not

- `anElement.hasAttribute(attName)`

Returns the value of a specified attribute on the element or `null`.

- `anElement.getAttribute(attName)`

Adds a new attribute or changes the value of an existing attribute on the specified element.

- `anElement.setAttribute(name, value)`

Removes an attribute from the specified element

- `anElement.removeAttribute(attrName)`

https://developer.mozilla.org/en-US/docs/Web/API/Element

# Element Attribute Manipulation

Creates a new attribute node, and returns it.

- `document.createAttribute(name)`

Represents the element's identifier, reflecting the id global attribute.

- `anElement.id`

An object representing the declarations of an element's style attributes.

- `anHTMLElement.style`

Returns the name of the element.

- `anElement.tagName`

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement
https://developer.mozilla.org/en-US/docs/Web/API/Element
https://developer.mozilla.org/en-US/docs/Web/API/Document

# Changing Style Programmatically

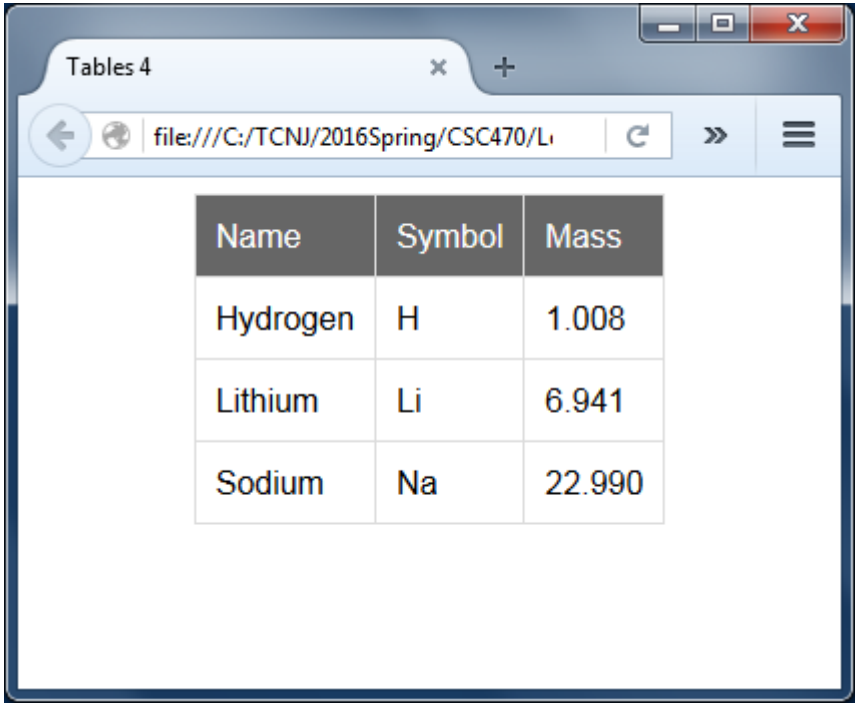
- An HTMLElement has a `style` property that may be used to change its style attributes
- Styles may be accessed using dot-notation or `[]`
- Style properties that contain dashes, such as `font-size`, cannot be accessed in JavaScript as properties. Must use `[]`
  - `anHTMLElement.style["font-size"]`
- Alternatively, you may use camel case
  - `anHTMLElement.style.fontSize`

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style

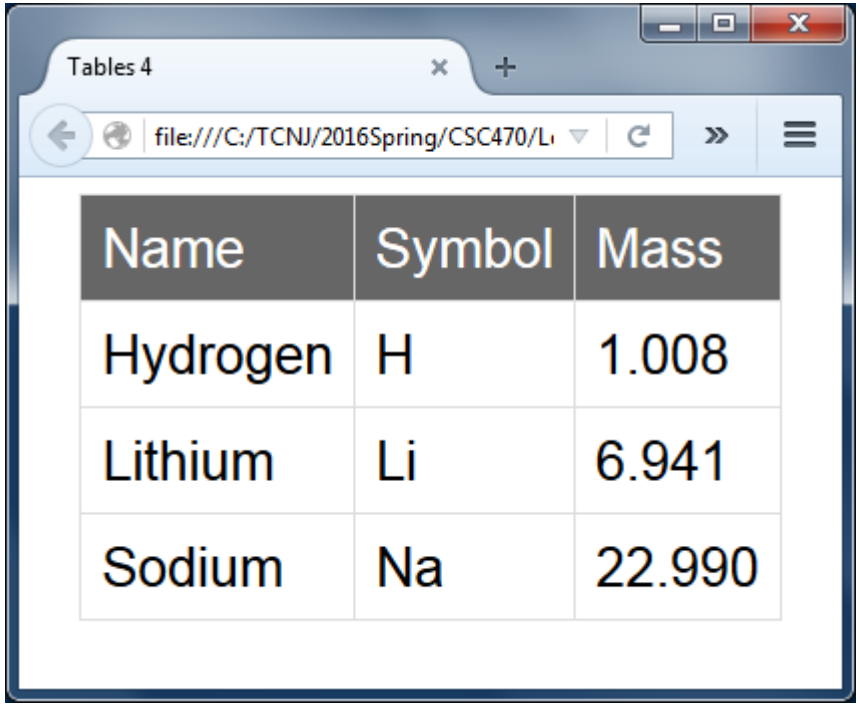
# Changing Style Programmatically

```
// Change the font size of the table
var setTableFontSize = function(size) {
  // Get table
  var tbl = document.getElementsByTagName('table')[0];

  // Change font size style
  tbl.style.fontSize = size;
};
```

```
setTableFontSize('20pt')
```




| Name | Symbol | Mass |
|---|---|---|
| Hydrogen | H | 1.008 |
| Lithium | Li | 6.941 |
| Sodium | Na | 22.990 |




| Name | Symbol | Mass |
|---|---|---|
| Hydrogen | H | 1.008 |
| Lithium | Li | 6.941 |
| Sodium | Na | 22.990 |

04/table4.html

# (Part of) The DOM Interface Hierarchy