What is HTML?

- A Markup Language for representing documents
  - text (data)
  - structure
  - appearance
  - functionality
- Designed for writing Web pages
- Traditionally rendered by Web browsers
  - Nowadays, also by other applications
Capabilities of HTML

- Content presentation
  - Structures, e.g., paragraphs, lists, tables, etc.
  - Decorations, e.g., fonts, images, etc.
- Declaration of meta information
  - e.g., the page title, language, etc.
- Linkage to other pages
  - i.e., attaching links to components
- Management of user input
  - e.g., searching, making reservations, ordering products
- Directions for browsers
  - e.g., refresh, redirect, caching control, etc.

A simple HTML page

```html
<html>
<head>
  <title>An HTML Page</title>
</head>
<body>
  <h1 id="hdr1">Hello World Wide Web!</h1>
</body>
</html>
```

Filename ends with .htm or .html
HTML Version History

- HTML 1.0 (first draft) – 1992
- HTML 2.0 (proposed standard) – September 1995
  
  *From this point on - W3C recommendations*

- HTML 3.2 – January 1997
  - added *tables, applets*, ...
- HTML 4.0 – December 1997
  - improved *tables, forms*, ...
- HTML 5.0 – January 2008
  - many new features *video, GPS location*, ...

---

HTML Version History (cont’d)

- HTML 4.01 – December 1999
  - Slightly different from 4.0
- XHTML 1.0 – January 2000
  - Reformulation of HTML 4.01 as an *XML* application
  - Stricter and cleaner syntax, formatting moved to CSS
- XHTML 1.1 – May 2001
  - “Modularization of XHTML”
**(X)HTML Support in Real Life**

- Non-standard / mixed HTML
  - Most of the time, will display more-or-less as expected, on most browsers, but its appearance may vary between browser types and between browser versions

- Even when using completely valid and standard HTML, **always check your pages on more than one browser type**
  - At least IE & some Mozilla

**Basic HTML Syntax**

- (X)HTML contains *text*, separated by *tags*
- Tags come in **pairs**: *opening* and *closing* tag
- Tags can have *attributes*, which have *values*

```
<html>
  <head><title>An HTML Page</title></head>
  <body>
    <h1 id="hdr1">Hello World Wide Web!</h1>
  </body>
</html>
```
Basic HTML Syntax (cont)

- An HTML page is surrounded by the `<html>` tag
- 2 Basic parts:
  - `<head>`: general information about the document (e.g., title – shown on the browser bar)
  - `<body>`: the content of the document

```
<html>
<head><title>An HTML Page</title></head>
<body>
  <h1 id="hdr1">Hello World Wide Web!</h1>
</body>
</html>
```

Actually a tree structure is created

XHTML Stricter Syntax

- Element and attr. names must be in lower case
- HTML allows any case combination, e.g., `<Body></BODY>`, which conveniently helped distinguish tags from the body text
- All tags must have corresponding closing tags
  - Use `<br/>` as a shorthand for `<br></br>`
- Elements should be properly nested
  - e.g., `<b>hello <em>world</em></b>` is illegal!
- Attribute values must be quoted
  - e.g., `<td rowspan="3">` and not `<td rowspan=3>`
Document Type Definitions

• It is good to specify which XHTML standard you are using
• Put a document type definition (DTD) at the first line of your file (before the html tag)
• For an example, see the next slide

Document Type Definitions (cont)

• XHTML - strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

• XHTML - transitional (loose)

```
<!DOCTYPE html PUBLIC
 "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

• XHTML - frameset (for using frames)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```  

Transition html→xhtml: allows some legacy formatting outside CSS
Basic Structures

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World Example</title>
<body>Hello World</body>
</html>

Links

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World Example</title>
<body><a href="anotherExample.html">Hello World</a></body>
</html>
What are Style Sheets

- A style sheet is a mechanism that allows to specify how HTML (/XHTML/XML) pages should look
- The style is specified by style rules
- The style rules appear either in the document or in external files, called style sheets
Style Sheets

- Usually, a file that ends with `.css`
- For example:
  - `i.a.cnn.net/cnn/.element/ssi/css/1.1/main.css` *(CNN)*
  - `http://www.technion.ac.il/styles/style.css` *(Technion)*
- To attach a style sheet to an HTML file, add `<link rel="stylesheet" type="text/css" href="css-file"/>` to the head

You can link to more than one css file

Style Sheets

- A file that is used for storing information about the way elements of HTML (or XML) should appear on the browser
- A style sheet increases the separation between content and presentation
  - Easier to generate large sites in which all the pages have the same style
  - It allows changing the look of many pages by changing a single file
  - May reduce network traffic
Without a style sheet (i.e., with the default style definitions)

`<html>
  <head><title>A Joke</title></head>
  <body>
    <div><img src="tomato.gif" alt="joke"/></div>
    <h1>A joke</h1>
    <p>A mama tomato, a papa tomato and a baby tomato are walking down the street. The baby tomato keeps falling behind so the papa tomato goes back, steps on the baby tomato and says, ketchup ("Catch-up!").</p>
  </body>
</html>`
Simple Example - with css

```html
<html>
<head>
<title>A Joke</title>
<link rel="stylesheet" type="text/css" href="joke.css"/>
</head>
<body>
<div><img src="tomato.gif" alt="joke"></div>
<h1>A joke</h1>
<p>A mama tomato, a papa tomato and a baby tomato are walking down the street. The baby tomato keeps falling behind so the papa tomato goes back, steps on the baby tomato and says, ketchup ("Catch-up!").</p>
</body>
</html>
```
Background: Style in Legacy HTML

- In traditional HTML, every HTML tag which supported changing its font/color/etc..., supported this using a different syntax
- Examples:

  ```html
  <body background="bg.jpg" vlink ="green">
    <center>Hello</center>
    <font size="3" color="red">Hey!</font>
    <strike>line-through</strike>
    <table border="1px">
  ```

Advantages of Style Sheets

- Separates content from style
  - Often require two kinds of skills; moreover, one is likely to change independently from the other
- Keeps HTML pages human-readable
- Reduces download time (*how?*)
- Allows to easily maintain a consistent appearance over a whole Web site (*why?*)
- A more flexible, unified way to specify style properties for different HTML elements (tables, paragraphs, etc...)
The Transition to Style Sheets

- Style Sheets provide new ways to accomplish tasks which were already possible
  - ... but recall the advantages from the previous slide
- Once style sheets were introduced, most of the legacy features with equivalent functionality were deprecated in HTML 4.01 and in Transitional XHTML and were removed in Strict XHTML

Style Rules (cont)

- A rule has the following form
  
  selector {declaration block}

- The selector determines when the rule is applied
- For example, the following rule applies to text that is inside a `<p>` tag

```
p {color: green; font-size: 1.5em; font-style: italic}
```

`em` is the current font-size of the element
Properties that CSS Controls

- **Style properties**
- **Layout properties**
  - There are many properties and many possible values
    - We will not cover all of them here
    - Look in the Web !!!

**Style Properties**
Our Examples

- We use the following HTML example:
  
  ```html
  This is <span>our example</span> for css.
  ```

- The `<span>` tag is used to group inline elements for formatting with styles
  
  - Extremely useful tag...

Font Properties

- Font properties: family, size, weight, style, variant, etc.

```css
span {
  font-family: courier;
  font-size: 130%;
  font-style: italic;
  font-weight: bold;
}
```

This is *our example* for css.
Text Properties

- Text properties: color, transform, decoration, ...

```css
span {
  color: #00cc00;
  text-decoration: line-through;
  text-transform: uppercase;
}
```

This is **OUR EXAMPLE** for css.

Background Properties

- Background properties: background-color, background-image, ...

```css
span {background-color: #00ff00}
```

This is **our example** for css.

```css
span {background-image: url("bg.gif");}
```

This is **our example** for css.
Layout Properties

Page Layout

- Each HTML element defines a **layer** (rectangular box) that is placed in some location on the page.
- Layers are **nested** with correspondence to the nesting of their elements.
Inline vs. Block Elements

- There are two types of elements:
  - **Block** elements: `p`, `ol`, `table`, `div`, `h1`, etc.
  - **Inline** elements: `b`, `i`, `a`, `span`, `cite`, etc.

- Layers of block elements are **separated** from their adjacent elements (i.e., a new line before and after), while inline elements are not.

- You can turn a block into an inline and vice-versa (highly useful for `img` elements, among others), using the `display` property, e.g., `h1 { display: inline }`

---

Positioning Elements

- Using CSS, you can define the position of an element inside its parent layer.

- For that, use the properties `position`, `left`, `right`, `top`

```
span {
    position: relative;
    left: 1cm;
    top: 1cm;
    color: #00cc00;
}
```

This is our example for css.
Position Types

- But 1cm left to what??
- For that, we have the position property
- Four position types are supported:
  - static: the default position
  - relative: relative to the static position
  - absolute: relative to the parent layer coordinates
  - fixed: relative to the window coordinates (remains at the same position regardless of scrolling)

Position Examples

```css
span {
  position: absolute;
  left: 1cm;
  top: 1cm;
  color: #00cc00;
}
```

This is for css.

```css
span {
  position: fixed;
  left: 1cm;
  top: 1cm;
  color: #00cc00;
}
```

This is for css.
Position Examples

```css
span {
  position: static;
  left: 1cm;
  top: 1cm;
  color: #00cc00;
}
```

This is the default position type

This is our example for css.

More Layout Properties

- Layer properties
  - `margin-top` (-bottom, -left, -right)
  - `padding-top` (-bottom, -left, -right)
  - `border-width` (-color, -style, ...)

- Text Layout
  - direction, word-spacing, white-space, letter-spacing, text-align, text-indent, ...
Length Units

- CSS has several types of length units:
  - em, ex: height of current fonts
    - ex is the height of “x”, em is the distance between bottoms of two subsequent lines
  - px, in, cm, mm, pt, pc: international units
  - %: ratio of parent's respective dimension
- A page should remain a \textit{proper layout} when fonts and/or windows \textit{are resized} (usually by the user)
  - Hence, do not assume anything about \textit{default sizes}

Selector Types
Several Kinds of Selectors

- Type Selectors
- Class Selectors
- ID Selectors
- Attribute Selectors
- Universal Selector
- Child Selectors
- Adjacent-Sibling Selectors
- Descendant Selectors
- Pseudo-Class Selectors
- Pseudo-Element Selectors

Type Selector

A *type selector* is the name of an element type

A type selector matches every instance of the element type

```html
li {color: red; font-size: 16px}
```

Matches:

```html
<ol>
  <li>An item</li>
  <li class="reditem">Another item</li>
</ol>
```
Universal Selector

- The *universal selector* matches every element
- The following rule means that all the text will have a size of **40px**

```
* {font-size: 40px }
```

Attribute Selectors

- `p[title]`
  - matches `p` when its `title` attribute is set to any value
- `p[title=intro]` or `p[title="intro"]` (the quotes are optional)
  - matches `p` when its `title` attribute is set to "intro"
- `p[class~=green]`
  - matches `p` when the `class` attribute value includes the word "green"
Class Selector

- A class selector is a selector of the form x.y
- It matches xs that have the class attribute with value y (i.e., it is a shorthand for x[\text{class}=y])

\begin{verbatim}
li.reditem {color: red}
\end{verbatim}

Matches:

<ol>
  <li> An item </li>
  <li class="reditem"> Another item </li>
</ol>

ID Selectors

- IDs are similar to classes, except that there can only be one element with a given ID in a document
- That is, an ID uniquely identifies an element

\begin{verbatim}
li#23 {color: red}
\end{verbatim}

Matches:

<ol>
  <li> An item </li>
  <li id="23"> Another item </li>
</ol>
Descendant/Child/Sibling Selector

- A **descendant selector** has the form $S_1S_2$ where $S_1$ and $S_2$ are (possible complex) selectors
- It matches all elements that
  - match $S_2$, and
  - are descendants (nested in any level in) elements that match $S_1$
- To match only immediate descendants (children), use a **Child Selector** $S_1 > S_2$
- To match $S_2$ immediately following $S_2$, use an **Adjacent-Sibling Selector** $S_1 + S_2$

An Example

```
.p em {color: blue}
```

**Matches:**

- This is `<em>`not blue`</em>`.
- `<p>`
  - This is `<em>`blue`</em>`
  - `<span>`<i>`and so is `<em>`this`</em>`</i>`</span>`.
- `</p>`

**What will this match?**

```
.head div>span em {color: blue}
```

Spring 2013  cs 236369  49

Spring 2013  cs 236369  50
Pseudo-Classes

- *Pseudo class* selectors are similar to class selectors, but they match *states* rather than class values
  - For example, a *link* can be in the states: *visited*, *active*, *mouse-over* (“*hover*”), etc.
  - Another example: a *text-area* can be *focused*

Examples of Rules for Pseudo-Classes

- `a:link` {color: blue}
- `a:visited` {color: purple}
- `a:hover` {font-size: 1.5em}
- `a:active` {color: red}
- `input[type=text]:focus` {background-color: yellow}

when typing a text into a text input box (meaningful with other input types as well)
Pseudo-Elements

- *Pseudo element* selectors select abstract elements which are not specified as elements in the source HTML
  - For example, to transform the first line of every `p` into uppercase, use:
    ```
P:first-line {text-transform: uppercase}
    ```
  - Why can’t this be faked by enclosing the first line with a span?
  - **Example 2:** `p.article:first-letter {color:#ff0000}`

Grouping Selectors

- We can group several declarations by specifying several selectors, separated by commas
  - For example, the following rule applies to all elements that match either `h1`, `p b`, or `h2[class=largehead]`
    ```
p b, h1, h2.largehead {font-size: 120%}
    ```
Inserting Style into a Page

Inline Styles

- In an inline style, the declaration block is the value of the attribute style of the element

```html
<p style="color: green; font-size: 1.5em; font-style: italic">
    This text will be shown in italic green and the size will be 1.5 times the current font size
</p>
```

- Almost every tag can have the style attribute
  - exceptional: base, head, html, meta, param, script, style and title
Document-Level Style

```html
<html>
<head>
  <style type="text/css">
    body {color: red; background: skyblue;}
    h1 { color: blue }
  </style>
</head>
<body>...
</body>
</html>
```

Imported Style Sheets

- The `@import` rule imports the style rules of another style sheet
- Usage: `@import url(file.css)`
- Several import rules may appear at the beginning of the style sheet
- Import rules can appear in embedded style sheets or in external style sheets
Imported Style Sheets

```css
@import url(general.css);
body { color: red; background:skyblue }
h1 { color: blue }
```

Why do we need the import command when we have the `<link>` tag?

Using `@import` in a css file, one can create style-sheets which are based on others…

Inheritance and Cascading
Inheritance of Properties

- Consider a property of an element that does not match any rule
- For some properties (inherited properties), the computed value of this property is inherited from the parent of the element
- For example, color, font and word-spacing are inherited
- Yet border, margin and padding are not!

An Example

- Given the rules:
  - body { font-size: 10pt }
  - h1 { font-size: 120% }

What will be the font size of the <em> element?

```html
<body>
  <h1>A <em>large</em> heading</h1>
</body>
```
Cascading of Styles

- CSS *merges* style rules from different places (inline, document-level, linked and defaults)
- Different places may have conflicting style rules
  - conflicts may even arise in a single source
- The process of merging (cascading) styles from different places determines which style rules have higher priority

Determining Property Values

- Suppose that we would like to determine the value of property *p* for element *e*
- Choose all declarations that have a selector that matches *e* and have the property *p*
- “Sort” all declarations according to *cascading order* (importance)
- Apply the rule of the first declaration in the cascading order
Cascading Order

The cascading order of declarations:

1. Primary sort: Importance of origin
2. Secondary sort: Specificity of selectors
3. Final sort: Order of appearance

Importance of Origin

- There are two origins of rules: author and browser (defaults / user customizations)

- `browser !important` rules
- `author !important` rules
- `author` rules
- `browser` rules

For example, you can add stylesheets to IE in the following way:
- Tools → internet options → Accessibility → User style sheet.
- Of course you can add `!important` browser rules this way as well…
Specificity of Selectors

**is rule in style attribute?**

**number of ID attributes**

**number of attributes and pseudo-classes**

**number of element names**

---

**An Example**

*Which is the most specific?*

- li {...}
- #x34y {...}
- ul ol li.red {...}

3 1 2
Client-Side Programming

- Certain parts of a Web application can be executed locally, in the client
- For example, some validity checks can be applied to the user’s input locally
- The user request is sent to the server only if the input is valid
- Java Script (**not part of Java!**) is an HTML-embedded scripting language for client-side programming

JavaScript

- JavaScript is a scripting language for generating dynamic HTML pages in the browser
- The script is written inside an HTML page and the browser runs the script and displays an ordinary HTML page
- There is some interaction of the script with the file system using **cookies**
- Cookies are small files that store personal information in the file system of the client
  - For example, a cookie may store your user name and password for accessing a particular site
Example 1

```html
<html>
<head><title>JS Example</title></head>
<body>
  <h2>Before the script</h2>
  <script type="text/javascript">
    document.write('<h1>In the script</h1>');
  </script>
  <h2>After the script</h2>
</body></html>
```

Example 2

```html
<html>
<head><title>JS Example</title></head>
<body>
  <h2>Before the script</h2>
  <h1>
    <script type="text/javascript">
      document.write(new Date().toLocaleString());
    </script>
  </h1>
  <h2>After the script</h2>
</body></html>
```
Example 3

```javascript
<h2>Hello and

```javascript
<i>```<script type="text/javascript">

hours = new Date().getHours();

if (hours < 10) { document.write("good morning") }
else {document.write("good day") }

</script></i>.

```javascript```
```
```
function gotoAd() {
    document.location.href = "http://" + adURLs[thisAd];
}
</script>
</head>
<body onload="cycleAds()" ...
<a href="javascript:gotoAd()">
<img name="adBanner" src="hosts/csail.gif" border="0" alt="Our sponsors"></a>