Today

HTML
  History: The Memex & Vannevar Bush
  Basic Syntax
  The Document Object Model (DOM)
  HTML 5 & Semantic Tags

CSS
  History: A Time Before CSS
  Basic Syntax
  Selectors
  In-Class Activity: CSS Dinner
  C is for Cookie Cascade
  Separation of Concerns
  Layout
    The Box Model
    Flex(ible) Box
Hypertext

HTML

Markup Language
https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/
Hello World
HTML is a markup language

<h1>Hello World</h1>
HTML is a markup language

<h1>Hello World</h1>
<h1>Hello World</h1>
<h1>Hello World</h1>
<h1>Hello World</h1>
Hello World

Opening Tag

Hello World

Closing Tag

Heading (level 1)
HTML can be nested

```html
<h1>Hello
<a href="http://mit.edu">World</a></h1>
```
HTML can be nested

```html
<h1>Hello
  <a href="http://mit.edu">World</a>
</h1>
```
<h1>Hello
<a href="http://mit.edu">World</a></h1>
<h1>Hello</h1>

<a href="http://mit.edu">World</a>

</h1>
**Opening Tag**

```
<h1>Hello</h1>
```

**Anchor**

```
<a href="http://mit.edu">World</a>
```

**Closing Tag**
<h1>Hello</h1>
<a href="http://mit.edu">World</a>
<h1>Hello</h1><a href="http://mit.edu">World</a>
The **Document Object Model (DOM)**

```
<h1>Hello
    <a href="http://mit.edu">World</a>
</h1>
```
The **Document Object Model (DOM)**

```html
<h1>Hello
   <a href="http://mit.edu">World</a>
</h1>
```
The **Document Object Model (DOM)**

```html
<h1>Hello
   <a href="http://mit.edu">World</a>
</h1>
```
The **Document Object Model (DOM)**

```
<h1>Hello
  <a href="http://mit.edu">World</a>
</h1>
```
The **Document Object Model (DOM)**

```html
<h1>Hello
  <a href="http://mit.edu">World</a>
</h1>
```

There's a one-to-one correspondence between HTML elements and DOM nodes.
The **Document Object Model (DOM)**

```html
<h1>Hello
    <a href="http://mit.edu">World</a>
</h1>
```

HTML attributes are stored as properties of DOM nodes.
HTML is forgiving
HTML is *forgiving*

```html
<h1>Hello
   <a href="http://mit.edu">World</a>
</h1>

<p>
   Thunder, thunder, thundercats, Ho!
</p>
```
HTML is **forgiving**

```html
<h1>Hello
  <a href="http://mit.edu">World</a>
</h1>

<p>Thunder, thunder, thundercats, Ho!</p>
```
HTML is forgiving but *boilerplate* helps find bugs and encourages *consistent rendering* between browsers/platforms.
<!DOCTYPE html>
<html>
<head>
    <title>My First Webpage!</title>
</head>
<body>
    <h1>Hello
        <a href="http://mit.edu">World</a>
    </h1>
    
    <p>
        Thunder, thunder, thundercats, Ho!
    </p>
    ...
</body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage!</title>
  </head>
  <body>
    <h1>Hello</h1>
    <a href="http://mit.edu">World</a>
    <p>Thunder, thunder, thundercats, Ho!</p>
  </body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage!</title>
  </head>
  <body>
    <h1>Hello <a href="http://mit.edu">World</a></h1>
    <p>Thunder, thunder, thundercats, Ho!</p>
  </body>
</html>
<!DOCTYPE html>
<html>
<head>
    <title>My First Webpage!</title>
</head>
<body>
    <h1>Hello
        <a href="http://mit.edu">World</a>
    </h1>
    <p>
        Thunder, thunder, thundercats, Ho!
    </p>
    ...
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <title>My First Webpage!</title>
</head>
<body>
  <h1>Hello
      <a href="http://mit.edu">World</a>
  </h1>
  
  <p>
    Thunder, thunder, thundercats, Ho!
  </p>
  ...
</body>
</html>
<!DOCTYPE html>
<html>
<head>
  <title>My First Webpage!</title>
</head>
<body>
  <h1>Hello
      <a href="http://mit.edu">World</a>
  </h1>
  
  <p>
      Thunder, thunder, thundercats, Ho!
  </p>
  
  ...
</body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage!</title>
  </head>
  <body>
    <h1>Hello
      <a href="http://mit.edu">World</a>
    </h1>
    <p>
      Thunder, thunder, thundercats, Ho!
    </p>
    ...
  </body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage!</title>
  </head>
  <body>
    <h1>Hello
      <a href="http://mit.edu">World</a>
    </h1>
    <p>
      Thunder, thunder, thundercats, Ho!
    </p>
    ...
  </body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage!</title>
  </head>
  <body>
    <h1>Hello</h1>
    <a href="http://mit.edu">World</a>
    
    <p>Thunder, thunder, thundercats, Ho!</p>
    ...
  </body>
</html>
<!DOCTYPE html>
<html>
    <head>
        <title>My First Webpage!</title>
    </head>
    <body>
        <h1>Hello
            <a href="http://mit.edu">World</a>
        </h1>

        <p>Thunder, thunder, thundercats, Ho!</p>

        ... 
    </body>
</html>
HTML describes **structure and content**

- `<h1>`–`<h6>`: Headings
- `<p>`: Paragraph
- `<span>`: Group elements on a single line
- `<div>`: Group elements across many lines
- `<article>`: Group elements in semantically meaningful ways
- `<section>`: Group elements in semantically meaningful ways
- `<nav>`: Group elements in semantically meaningful ways
- `<img src=""/>`: Multimedia
- `<video src=""/>`: Multimedia
- `<audio src=""/>`: Multimedia
- `<ul>`: Unordered List (bullet points)
- `<li>`: Unordered List (bullet points)
- `<ol>`: Ordered (numbered) List
- `<li>`: Ordered (numbered) List
- `<table>`: Tables with rows and cells
- `<tr>`: Tables with rows and cells
- `<td>`: Tables with rows and cells
- `<form>`: Forms with user-submittable input (e.g., textboxes, sliders, checkboxes, dropdowns, ...)
The Year is 1993
The Year is 1993

HTML v1.0 is first published
The Year is 1993

HTML v1.0 is first published

... but doesn't offer a way to style content
Welcome to the electronic Telektronikk. The electronic version was available through the web one week before paper, and has enjoyed several corrections after the ink had dried. It has also received honorable mention at the best of Web94. Due to high demand, we are no longer able to send out complimentary paper copies, but help yourself to the electronic version - of which there are unlimited copies.

- Guest editorial by Håkon W Lie
- Windows into Cyberspace by Håkon W Lie
- Altruism and benefit in Cyberspace by Børre Ludvigsen
- Listen to Internet by Per E Dybkvik
- Telecommunications and CD-ROM - friends or foes? by Erling Maartmann-Moe
- The digital video album: On the merging of media types in multimedia by Gunnar Liestøl
- Hypertext reading as practical action: notes on technology, objectivation and knowledge by Terje Rasmussen
- Information exchange in MultiTorg by Dag Solvoll, Geir Ivarsøy, Håkon W Lie, and Per E Dybkvik
- Media streams: an iconic visual language for video annotation by Marc Davis
- SCREAM: Screen-based navigation in voice messages by Håkon W Lie, Per E Dybkvik, and Jan Rygh
- Telecommunications and social interaction - Social constructions in virtual space by Ola Ødegård
- Distributed Virtual Reality, applications for education, entertainment and industry by Carl E Loeffler
- Coordination: challenge of the nineties: Multimedia as a coordination technology by Per M Schiefløe and Tor G Syvertsen
- An informal requirements analysis of Norwegian public administration relative to CSCW by Pål Sørgaard
- International Information Infrastructure: social and policy considerations by David Hakken
Welcome to the electronic Telektronikk. The electronic version was available through the web one week before paper, and after the ink had dried. It has also received honorable mention at the best of Web94. Due to high demand, we are no longer making paper copies, but help yourself to the electronic version - of which there are unlimited copies.

- Guest editorial by Håkon W Lie
- Windows into Cyberspace by Håkon W Lie
- Altruism and benefit in Cyberspace by Børre Ludvigsen
- Listen to Internet by Per E Dybvik
- Telecommunications and CD-ROM - friends or foes? by Erling Maartmann-Moe
- The digital video album: On the merging of media types in multimedia by Gunnar Liestøl
- Hypertext reading as practical action: notes on technology, objectivation and knowledge by Terje Rasmussen
- Information exchange in MultiTorg by Dag Solvoll, Geir Ivarssøy, Håkon W Lie, and Per E Dybvik
- Media streams: an iconic visual language for video annotation by Marc Davis
- SCREAM: Screen-based navigation in voice messages by Håkon W Lie, Per E Dybvik, and Jan Rygh
- Telecommunications and social interaction - Social constructions in virtual space by Ola Ødegård
- Distributed Virtual Reality: applications for education, entertainment and industry by Carl E Loeffler
- Coordination: challenge of the nineties: Multimedia as a coordination technology by Per M Schiefloe and Tor G Syvertsen
- An informal requirements analysis of Norwegian public administration relative to CSCW by Pål Søgaard
- International Information Infrastructure: social and policy considerations by David Hakken
Welcome to the electronic Telektronikk. The electronic version was available through the web one week before paper, and after the ink had dried. It has also received honorable mention at the best of Web'94. Due to high demand, we are no longer printing the paper copies, but help yourself to the electronic version - of which there are unlimited copies.

- Guest editorial by Håkon W Lie
- Windows into Cyberspace by Håkon W Lie
- Altruism and benefit in Cyberspace by Børre Ludvigsen
- Listen to Internet by Per E Dybvik
- Telecommunications and CD-ROM - friends or foes? by Erling Maartmann-Moe
- The digital video album: On the merging of media types in multimedia by Gunnar Liestøl
- Hypertext reading as practical action: notes on technology, objectification and knowledge by Terje Rasmussen
- Information exchange in MultiTorg by Dag Solvoll, Geir Ivarsøy, Håkon W Lie, and Per E Dybvik
- Media streams: an iconic visual language for video annotation by Marc Davis
- SCREAM: Screen-based navigation in voice messages by Håkon W Lie, Per E Dybvik, and Jan Rygh
- Telecommunications and social interaction - Social constructs in virtual space by Ola Ødegård
- Distributed Virtual Reality. Applications for education, entertainment and industry by Carl E Loeffler
- Coordination: challenge of the nineties. Multimedia as a coordination technology by Per M Schieffoe and Tor G Syvertsen
- An informal requirements analysis of Norwegian national administration relative to CSCW by Pål Sørgaard
- International Information Infrastructure: social and policy considerations by David Hakken
Welcome to the electronic Telektronikk. The electronic version was available through the web one week before paper, and after the ink had dried. It has also received honorable mention at the best of Web94. Due to high demand, we are no longer distributing paper copies, but help yourself to the electronic version - of which there are unlimited copies.

- Guest editorial by Håkon W Lie
- Windows into Cyberspace by Håkon W Lie
- Altruism and benefit in Cyberspace by Børre Ludvigsen
- Listen to Internet by Per E Dybvik
- Telecommunications and CD-ROM - friends or foes? by Erling Maartmann-Moe
- The digital video album: On the merging of media types in multimedia by Gunnar Liestøl
- Hypertext reading as practical action: notes on technology, objectivation and knowledge by Terje Rasmussen
- Information exchange in MultiTorg by Dag Solvoll, Geir Ivarsøy, Håkon W Lie, and Per E Dybvik
- Media streams: an iconic visual language for video annotation by Marc Davis
- SCREAM: Screen-based navigation in voice messages by Håkon W Lie, Per E Dybvik, and Jan Rygh
- Telecommunications and social interaction - Social constructions in virtual space by Ola Ødegård
- Distributed Virtual Reality, applications for education, entertainment and industry by Carl E Loeffler
- Coordination: challenge of the nineties: Multimedia as a coordination technology by Per M Schieffloe and Tor G Syvertsen
- An informal requirements analysis of Norwegian public administration relative to CSCW by Pål Sørgaard
- International Information Infrastructure: social and policy considerations by David Hakken

howcome
The Year is 1997
The Year is 1997

HTML v3.1 is released
The Year is 1997

HTML v3.1 is released

with style tags (e.g., `<font>`, `<b>`, `<i>`, `<u>`, ...) and attributes (e.g., `bgcolor`, `align`, `border`, ...)
<html>
<head>
<title>My First Webpage!</title>
</head>
<body>
<h1>Hello <a href="http://mit.edu">World</a></h1>

<p>Thunder, thunder, thundercats, Ho!</p>

...</p>
</body>
</html>
<html>
<head>
  <title>My First Webpage!</title>
</head>
<body bgcolor="cyan" link="red">
  <h1>Hello
      <a href="http://mit.edu">World</a>
  </h1>

  <p>
      Thunder, thunder, thundercats, Ho!
  </p>

  ...
</body>
</html>
<html>
<head>
    <title>My First Webpage!</title>
</head>
<body bgcolor="cyan" link="red">
    <h1 align="center">Hello
        <a href="http://mit.edu">World</a>
    </h1>

    <p>
        Thunder, thunder, thundercats, Ho!
    </p>

    ...
</body>
</html>
<html>
<head>
  <title>My First Webpage!</title>
</head>
<body bgcolor="cyan" link="red">
  <h1 align="center">Hello
    <a href="http://mit.edu">World</a>
  </h1>

  <p><font color="orange"><i>
    Thunder, thunder, thundercats, Ho!
  </i></font></p>
  ...
</body>
</html>
<html>
<head>
  <title>My First Webpage!</title>
</head>
<body bgcolor="cyan" link="red">
  <h1 align="center">Hello
    <a href="http://mit.edu">World</a>
  </h1>
  <p><font color="orange"><i>
    Thunder, thunder, thundercats, Ho!
  </i></font></p>
  ...
</body>
</html>

- Inconsistent ways of styling elements (tags vs. attributes).
- Difficult to reapply styles across elements.
- Difficult to redesign pages.
- HTML is noisier and more complex: mixing content and presentation.
CSS
Cascading CSS
Cascading CSS Style Sheets
A CSS rule *describes presentation*

```css
p {
  font-family: Helvetica, Arial;
  font-size: 16px;
  color: #333333;
}
```
A CSS rule *describes presentation*

**Selector**: which elements (e.g., `<p>`) the rule applies to.

```css
p {
  font-family: Helvetica, Arial;
  font-size: 16px;
  color: #333333;
}
```
A CSS rule *describes presentation*

**Selector**: which elements (e.g., `<p>`) the rule applies to.

```css
p {
  font-family: Helvetica, Arial;
  font-size: 16px;
  color: #333333;
}
```
A CSS rule describes presentation

**Selector**: which elements (e.g., `<p>`) the rule applies to.

```css
p {
  font-family: Helvetica, Arial;
  font-size: 16px;
  color: #333333;
}
```

**Declaration**

**Property**
A CSS rule describes presentation

**Selector**: which elements (e.g., `<p>`) the rule applies to.

```css
p {
font-family: Helvetica, Arial;
font-size: 16px;
color: #333333;
}
```

**Declaration**

- `font-family`: Helvetica, Arial
- `font-size`: 16px
- `color`: #333333
CSS selectors *identify* HTML elements
CSS selectors identify HTML elements

* Universal selector: selects all elements.
CSS selectors * identify HTML elements

* Universal selector: selects all elements.

**element** Selects elements by tag (e.g., p, div, h1, ul, li)
**CSS selectors** identify HTML elements

- * Universal selector: selects *all* elements.
- `element` Selects elements by *tag* (e.g., `p`, `div`, `h1`, `ul`, `li`)
- `[attr="val"]` Selects elements by *attribute* name (and value)
CSS selectors identify HTML elements

* Universal selector: selects all elements.

**element** Selects elements by tag (e.g., *p*, *div*, *h1*, *ul*, *li*)

*[attr="val"]* Selects elements by attribute name (and value) (e.g., *a*[href="http://mit.edu"] selects all links to MIT)
CSS selectors identify HTML elements

* Universal selector: selects all elements.

**element** Selects elements by tag (e.g., p, div, h1, ul, li)

```
[attr="val"]
```
Selects elements by attribute name (and value)
(e.g., a[href="http://mit.edu"] selects all links to MIT)

```
#id
```
Selects elements by their id attribute
<table>
<thead>
<tr>
<th>CSS selector</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Universal selector: selects all elements.</td>
</tr>
<tr>
<td>element</td>
<td>Selects elements by tag (e.g., <code>p</code>, <code>div</code>, <code>h1</code>, <code>ul</code>, <code>li</code>)</td>
</tr>
<tr>
<td><code>[attr=&quot;val&quot;]</code></td>
<td>Selects elements by attribute name (and value) (e.g., <code>a[href=&quot;http://mit.edu&quot;]</code> selects all links to MIT)</td>
</tr>
<tr>
<td>#id</td>
<td>Selects elements by their <code>id</code> attribute (e.g., <code>#header</code> would select <code>&lt;div id=&quot;header&quot;&gt;&lt;/div&gt;</code>)</td>
</tr>
</tbody>
</table>
**CSS selectors** *identify HTML elements*

* Universal selector: selects *all* elements.

**element** Selects elements by *tag* (e.g., `p`, `div`, `h1`, `ul`, `li`)

`[attr="val"]` Selects elements by *attribute* name (and value) (e.g., `a[href="http://mit.edu"]` selects all links to MIT)

`#id` Selects elements by their *id* attribute (e.g., `#header` would select `<div id="header"></div>`)  

.`class` Selects elements by their *class* attribute
CSS selectors **identify HTML elements**

* Universal selector: selects *all* elements.

**element** Selects elements by *tag* (e.g., `p`, `div`, `h1`, `ul`, `li`)

```
[attr="val"]
```
Selects elements by *attribute* name (and value)
(e.g., `a[href="http://mit.edu"]` selects all links to MIT)

**#id** Selects elements by their *id* attribute
(e.g., `#header` would select `<div id="header"></div>`)

**.class** Selects elements by their *class* attribute
(e.g., `.photo` would select `<div class="photo"></div>`)
CSS selectors identify HTML elements

* Universal selector: selects all elements.

element Selects elements by tag (e.g., p, div, h1, ul, li)

[attr="val"] Selects elements by attribute name (and value) (e.g., a[href="http://mit.edu"] selects all links to MIT)

#id Selects elements by their id attribute (e.g., #header would select <div id="header"></div>)

.class Selects elements by their class attribute (e.g., .photo would select <div class="photo"></div>)

:hover Pseudo-classes select special/interactive state

:checked

:focus, etc.
CSS can be *combined*
CSS can be *combined*

`a[href="http://mit.edu"].home` Selects all links to MIT with a class `home`. 

62
CSS can be combined

`a[href="http://mit.edu"][.home` Selects all links to MIT with a class `home`.

`div.headline, .photo` A `comma` specifies a list of individual selections.
CSS can be **combined**

`a[href="http://mit.edu"] .home`  Selects all links to MIT with a class `home`.

`div.headline, .photo`  A *comma* specifies a list of individual selections.
  (e.g., selecting all elements with a `photo` class, and `divs` with a `headline` class)
CSS can be *combined*

```html
div.headline .photo
```
Select *all* nested (or descendent) elements.
CSS can be combined

div.headline .photo
Select all nested (or descendental) elements.

div.headline > .photo
Select immediate children.
CSS can be combined

\[ \text{img} \sim \text{p} \]
Selects subsequent siblings
CSS can be combined

`img ~ p`
Selects subsequent siblings

`img + p`
Select immediate siblings.
In-Class Activity: Practice CSS Selectors

CSS Dinner
https://flukeout.github.io/

For credit, take a screenshot when time is called and submit it to
https://www.yellkey.com/drop
Kombucha brunch intelligentsia
Which rule wins?

```html
<body>
  <div class="headline">
    <h1>
      Kombucha brunch intelligentsia
    </h1>
  </div>
</body>
```
Which rule wins?

<body>
  <div class="headline">
    <h1>
      Kombucha brunch intelligentsia
    </h1>
  </div>
</body>

body {
  color: red;
}
Which rule wins?

```html
<body>
  <div class="headline">
    <h1>
      Kombucha brunch intelligentsia
    </h1>
  </div>
</body>
```

1. body {
   color: red;
}

2. div.headline {
   color: green;
}
Which rule wins?

```html
<body>
  <div class="headline">
    <h1>
      Kombucha brunch intelligentsia
    </h1>
  </div>
</body>
```

1. `body` {
   color: red;
}

2. `div.headline` {
   color: green;
}

3. `div.headline h1` {
   color: blue;
}
CSS: The *cascade* determines rule priority
CSS: The *cascade* determines rule priority

1. The `!important` flag.
CSS: The *cascade* determines rule priority

1. The `!important` flag.

```css
body {
  color: red !important;
}

div.headline {
  color: green;
}

div.headline h1 {
  color: blue;
}
```
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:
   
   +1000 if styles specified inline (i.e., inside an element's `style` attribute).
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:
   - +1000 if styles specified inline (i.e., inside an element's `style` attribute).
   - +100 for every `#id`
CSS: The *cascade* determines rule priority

1. The !important flag.

2. How specific the selector is, scored by:
   - +1000 if styles specified inline (i.e., inside an element's `style` attribute).
   - +100 for every `#id`
   - +10 for every `.class`, `[attr]`, `:pseudo`
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:
   - +1000 if styles specified inline (i.e., inside an element's `style` attribute).
   - +100 for every `#id`
   - +10 for every `.class, [attr], :pseudo`
   - +1 for every `element`
+1000 if styles specified inline (i.e., inside an element's style attribute).
+100 for every #id
+10 for every .class, [attr], :pseudo
+1 for every element

```html
<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
```

```css
body {
  color: red;
}

div.headline {
  color: green;
}

div.headline h1 {
  color: blue;
}
```
+1000 if styles specified inline (i.e., inside an element's `style` attribute).
+100 for every `#id`
+10 for every `.class`, `[attr]`, `:pseudo`
+1 for every `element`

```html
<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
```
+1000 if styles specified inline (i.e., inside an element's `style` attribute).
+100 for every `id`
+10 for every `.class`, `[attr]`, `:pseudo`
+1 for every element

```
<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
```
+1000 if styles specified inline (i.e., inside an element's style attribute).
+100 for every #id
+10 for every .class, [attr], :pseudo
+1 for every element

```html
<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
```

```css
body {
  color: red;
}

+1 = 1

div.headline {
  color: green;
}

div.headline h1 {
  color: blue;
}
```
+1000 if styles specified inline (i.e., inside an element's style attribute).
+100 for every #id
+10 for every .class, [attr], :pseudo
+1 for every element

+1 = 1
body {
  color: red;
}
+1 +10 = 11
div.headline {
  color: green;
}
div.headline h1 {
  color: blue;
}

<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
+1000 if styles specified inline (i.e., inside an element's style attribute).
+100 for every #id
+10 for every .class, [attr], :pseudo
+1 for every element

```
<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>
```

+1 = 1
```
body {
  color: red;
}
```
```
+1 +10 = 11
```
```
div.headline {
  color: green;
}
```
```
+1 +10
```
```
div.headline h1 {
  color: blue;
}
```
+1000 if styles specified inline (i.e., inside an element's style attribute).
+100 for every #id
+10 for every .class, [attr], :pseudo
+1 for every element

<body>
  <div class="headline">
    Kombucha brunch intelligentsia
  </div>
</body>

+1 = 1
body {
  color: red;
}
+1 +10 = 11
div.headline {
  color: green;
}
+1 +10 +1 = 12
div.headline h1 {
  color: blue;
}
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:

   + **1000** if styles specified inline (i.e., inside an element's `style` attribute).
   + **100** for every `#id`
   + **10** for every `.class`, `[attr]`, `:pseudo`
   + **1** for every `element`
CSS: The *cascade* determines rule priority

1. The `!important` flag.

2. How specific the selector is, scored by:
   
   +1000  if styles specified inline (i.e., inside an element's `style` attribute).
   
   +100   for every `#id`
   
   +10    for every `.class`, `[attr]`, `:pseudo`
   
   +1     for every `element`

3. Source code order: rules defined later in a stylesheet have higher priority over those defined earlier.
Embedding CSS in HTML

```html
<html>
<head>
  <link rel="stylesheet" href="photos.css">
  <style>
    div.headline { color: green; }
  </style>
</head>

<body>
  <h1>Hello</h1>
</body>
</html>
```
Embedding CSS in HTML

```
<html>
  <head>
    <link rel="stylesheet" href="photos.css"> 
    <style>
      div.headline { color: green; }
    </style>
  </head>
  <body>
    <h1>Hello</h1>
  </body>
</html>
```
Embedding CSS in HTML

```html
<html>
<head>
  <link rel="stylesheet" href="photos.css">
  <style>
    div.headline { color: green; }
  </style>
</head>

<body>
  <h1>Hello</h1>
</body>
</html>
```

External styles

Internal styles
Embedding CSS in HTML

```html
<html>
<head>
  <link rel="stylesheet" href="photos.css">
  <style>
    div.headline { color: green; }
  </style>
</head>

<body>
  <h1 style="color: red;">Hello</h1>
</body>
</html>
```
Embedding CSS in HTML

```html
<html>
<head>
  <link rel="stylesheet" href="photos.css">  
  <style>
    div.headline { color: green; }
  </style>
</head>
<body>
  <h1 style="color: red;">Hello</h1>
</body>
</html>
```

- **External styles**
- **Internal styles**
- **Inline styles**
Embedding CSS in HTML

```html
<html>
  <head>
    <link rel="stylesheet" href="photos.css">
    <style>
      div.headline { color: green; }
    </style>
  </head>
  <body>
    <h1 style="color: red;">Hello</h1>
  </body>
</html>
```
HTML + CSS: A better *separation of concerns*

```html
<html>
<head>
    <link rel="stylesheet" href="photos.css">
</head>

<body>
    <h1>Hello</h1>

    <p>Kombucha brunch intelligentsia</p>
</body>
</html>
```
HTML + CSS: A better separation of concerns

• Easier to reuse styles to maintain visual consistency across pages.

<html>
<head>
  <link rel="stylesheet" href="photos.css">
</head>

<body>
  <h1>Hello</h1>
  
  <p>Kombucha brunch intelligentsia</p>
</body>
</html>
HTML + CSS: A better separation of concerns

- Easier to reuse styles to maintain visual consistency across pages.
- Easier to redesign pages.

```html
<html>
<head>
  <link rel="stylesheet" href="photos.css">
</head>

<body>
  <h1>Hello</h1>
  <p>Kombucha brunch intelligentsia</p>
</body>
</html>
```
HTML + CSS: A better separation of concerns

- Easier to reuse styles to maintain visual consistency across pages.
- Easier to redesign pages.
- Smaller page sizes and faster rendering.
HTML + CSS: A better *separation of concerns*

- Easier to **reuse** styles to maintain visual consistency across pages.
- Easier to **redesign** pages.
- **Smaller** page sizes and **faster rendering**.
- More **accessible** for assistive technologies (e.g., screen readers) and search engines.
HTML + CSS: A better *separation of concerns*

```html
<html>
  <head>
    <link rel="stylesheet" href="photos.css">
  </head>
  <body>
    <h1>Hello</h1>
    <p>Kombucha brunch intelligentsia</p>
  </body>
</html>
```

- Easier to **reuse** styles to maintain visual consistency across pages.
- Easier to **redesign** pages.
- Smaller page sizes and **faster rendering**.
- More **accessible** for assistive technologies (e.g., screen readers) and search engines.
- **Retargeting** for different devices (e.g., mobile, tablet) and use-cases.
Layout
Element size is calculated with the box model
Element size is calculated with the box model

Content
(Text / Image / ...)
Element size is calculated with the box model.

The width and height properties determine the size of the content.

Content
(Text / Image / ...)

width
height

auto by default: browser calculates a size to fit the content.
Element size is calculated with the box model. The width and height properties determine the size of the content. By default, auto is used: the browser calculates a size to fit the content.

- **width**
- **height**

Content
(Text / Image / ...)

Height and width arrows above and below the content box.
Element size is calculated with the **box model**

**padding** adds whitespace around the content.
Element size is calculated with the box model.

Padding

- padding-top
- padding-left
- padding-right
- padding-bottom

Content

Width

Height

Padding adds whitespace around the content.
Element size is calculated with the *box model*.
Element size is calculated with the **box model**
Element size is calculated with the box model.

Margin

Padding

Content

Border

margin adds whitespace between elements.
Element size is calculated with the box model. The box model consists of:

- **Margin**: Adds whitespace between elements.
- **Padding**: Internally adds space around the content.
- **Content**: The actual content of the element.
- **Border**: The border around the content.

The margin is specified using `margin-top`, `margin-right`, `margin-bottom`, and `margin-left` properties.
Element size is calculated with the box model

Full size = width/height
Full size is + padding
Full size is + border
Full size is + margin
Element size is calculated with the **box model**

Two types of units:

- **absolute** (*px*, *pt*, *cm*, ...)  
  An exact measurement, that won't respect any defaults a user has set in their browser.

- **relative** (*em*, %, *vw*, ...)  
  Calculated with respect to some base reference (e.g., *em* is relative to the element's **font-size**).
Flex(ible) Box: Redistributing Free Space in 1D
Flex(ible) Box: Redistributing Free Space in 1D
Flex(ible) Box: Redistributing Free Space in 1D

```css
.container {
  display: flex;
}
```
Flex(ible) Box: Redistributing Free Space in 1D

```css
.container {
    display: flex;
}
```

Ratios to control how an `.item` grows or shrinks when the size of the `.container` changes.
**flex-grow:** Ratio to stretch items to fill free space

```
.container {
  display: flex;
}
```
**flex-grow**: Ratio to stretch items to fill free space

```
#one .item
#two .item
#three .item

.container {
  display: flex;
}
```

How to distribute this free space to .items?
**flex-grow**: Ratio to stretch items to fill free space

How to distribute this free space to `.item`s?

```css
.container {
  display: flex;
}

.item {
  flex-grow: 1;
}
```
**flex-grow**: Ratio to stretch items to fill free space

```
.container {
  display: flex;
}

.item {
  flex-grow: 1;
}
```

How to distribute this free space to `.item`s?

"Evenly distribute the space."
**flex-grow:** Ratio to stretch items to fill free space

```
.container {
  display: flex;
}
.item {
  flex-grow: 1;
}
```

"Evenly distribute the space."
**flex-grow**: Ratio to stretch items to fill free space

The `flex-grow` property is used to specify the ratio by which an item grows to fill the available space.

```
.container {
  display: flex;
}
.item {
  flex-grow: 1;
}
#one {
  flex-grow: 2;
}
```

How to distribute this free space to `.item`s?
flex-grow: Ratio to stretch items to fill free space

```
.container {
  display: flex;
}

.item {
  flex-grow: 1;
}

#one {
  flex-grow: 2;
}
```
flex-shrink: Ratio to compress items

.container {
  display: flex;
}

#one .item

#two .item

#three .item
flex-shrink: Ratio to compress items

How to account for this overflow space?
**flex-shrink**: Ratio to compress items

```css
.container {
  display: flex;
}

.item {
  flex-shrink: 1;
}
```

How to account for this overflow space?
**flex-shrink**: Ratio to compress items

```
.container {
  display: flex;
}
.item {
  flex-shrink: 1;
}
```

How to account for this overflow space?

"Evenly distribute negative free space aka shrink items proportionately."
**flex-shrink**: Ratio to compress items

```
.container {
  display: flex;
}

.item {
  flex-shrink: 1;
}
```

"Evenly distribute negative free space aka shrink items proportionately."
**flex-shrink**: Ratio to compress items

Evenly distribute negative free space aka shrink items proportionately.

*In practice, takes into account how much an item can shrink, so that e.g. a small item won’t shrink to zero before a larger item has been noticeably reduced.*

"Evenly distribute negative free space aka shrink items proportionately."
**flex-basis**: Initial size before growth/shrinkage

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>auto</code></td>
<td>The default size of the content</td>
</tr>
<tr>
<td><code>0</code></td>
<td>Items start at 0, so they are equally sized with flexbox.</td>
</tr>
<tr>
<td><code>width</code></td>
<td>(e.g., <code>200px</code>, <code>0.5vw</code>)</td>
</tr>
</tbody>
</table>
flex: Recommended shorthand
**flex**: Recommended shorthand

```
flex: 5;  // flex-grow: 5;
flex-shrink: 1;
flex-basis: 0;
```
**flex**: Recommended shorthand

- `flex: 5;` → `flex-grow: 5; flex-shrink: 1; flex-basis: 0;`
- `flex: 5 30px;` → `flex-grow: 5; flex-shrink: 1; flex-basis: 30px;`
**flex**: Recommended shorthand

```css
flex: 5;  // flex-grow: 5; flex-shrink: 1; flex-basis: 0;
flex: 5 30px;  // flex-grow: 5; flex-shrink: 1; flex-basis: 30px;
flex: 5 2 30px;  // flex-grow: 5; flex-shrink: 2; flex-basis: 30px;
```
The **Web Inspector**

---

**Hello World**

Thunder, thunder, thunderscats, Ho!

---

**Hello World**

Thunder, thunder, thunderscats, Ho!
Assignment 1: Personal Website

Updated: 05 Sep 2018

6.170 Assignment 1: Personal Website

Due: 11 Sep 2018 @ 11:59 a.m. — Submission Form

Since this is the first assignment, please take a look at this course's collaboration and grading policy in the General Information before beginning.

Overview

Your challenge in this assignment is to build a website using HTML and CSS. Your website should be purely informational with no dynamic or interactive elements.

The content of the website is up to you. Our recommendation is to create a personal website that presents your biographical information, interests, projects you have completed, and anything else you want the world to know about you. A personal website can serve as your digital identity and a portfolio of your work. Alternatively, you can create a website on a topic of your choosing such as a city you have enjoyed visiting or a historical figure you admire.

Whatever you choose, you should present your content in a clean and organized manner that takes advantage of the features of HTML and CSS. Although we do not cover them deeply in this class, we expect you to apply the good software engineering practices taught in this class's prerequisites (i.e. 6.031) to all assignments in this class, including this one.

Objectives