CSS: Cascading Style Sheets

BASICS, SELECTORS, PAGE LAYOUT AND RESPONSIVE DESIGN

POLITECNICO DI TORINO

Laura Farinetti - DAUIN

e-Lite

Summary

- Introduction
- CSS syntax
- CSS selectors
- CSS box model
- Page layout with CSS
- CSS media queries and responsive web design

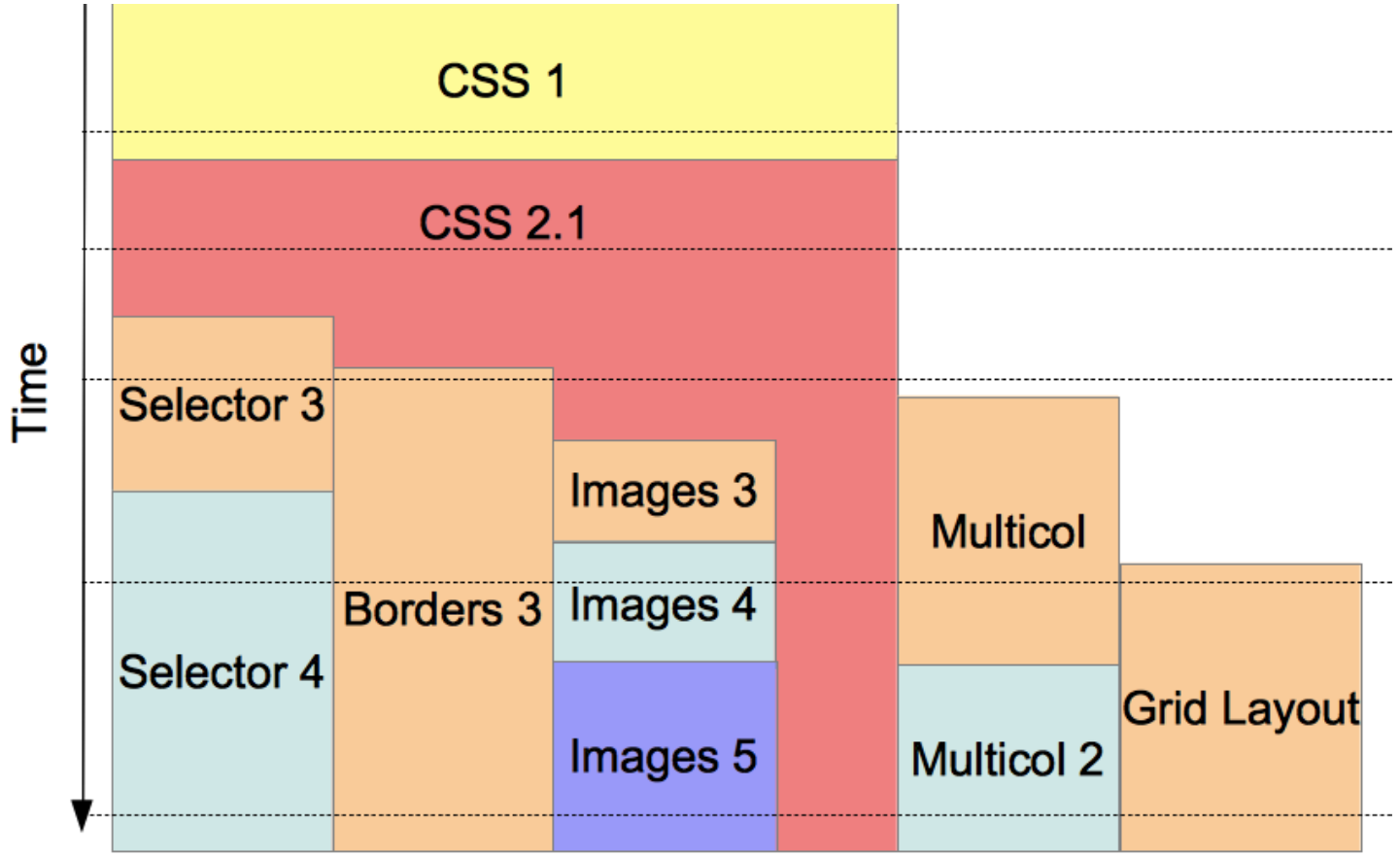
Cascading Style Sheets

- CSS: Cascading Style Sheet
- CSS 1: W3C recommendation (17 Dec 1996)
- CSS 2.1: W3C Recommendation (7 June 2011)
- CSS 3: different stages (REC, PR, CR, WD)
  - see http://www.w3.org/Style/CSS/current-work
- Resources:
  - CSS 2.1 standard, http://www.w3.org/TR/CSS21/
  - W3C CSS Tutorial, http://www.w3.org/Style/Examples/011/firstcss

CSS level 3 (CSS3)

- Major change: introduction of modules
- Advantage of modules: they (supposedly) allow the specification to be completed and approved more quickly, because segments are completed and approved in chunks
  - This also allows browser and user-agent manufacturers to support sections of the specification but keep their code bloat to a minimum by only supporting those modules that make sense

Overview

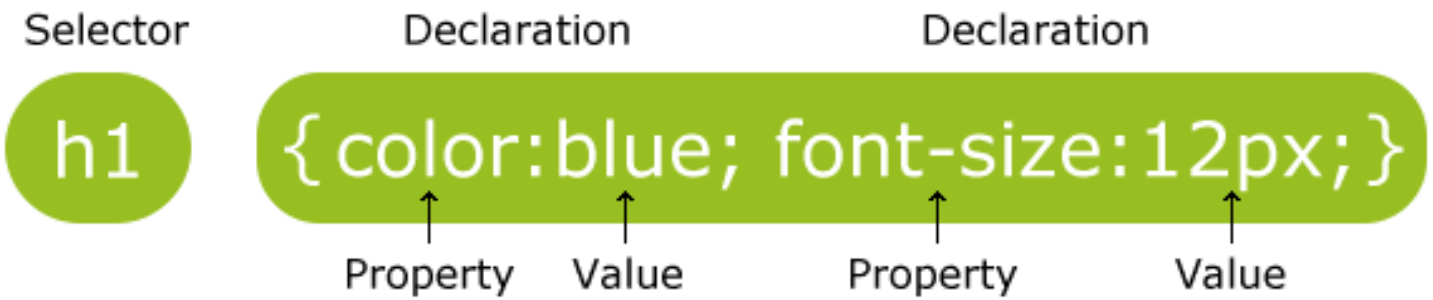


https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

CSS SYNTAX

CSS Syntax

- CSS is based on rules
- A rule is a statement about one stylistic aspect of one or more HTML element
- A style sheet is a set of one or more rules that apply to an HTML document



HTML structure

```
<html lang="en">
<head>
    <title>Trickier nesting, still</title>
</head>
<body>
    <div>
        <div>
            <table>
                <tr><th>Steps</th><th>Processes</th></tr>
                <tr><td>1</td><td>Figure out the <em>root element</em>.</td></tr>
                <tr><td>2</td><td>Deal with the <span>head</span> first as it's
                                usually easy.</td></tr>
                <tr><td>3</td><td>Work through the <span>body</span>. Just
                                <em>take your time</em>.</td></tr>
            </table>
        </div>
        <div>
            This link is <em>not</em> active, but it it were, the answer to this
            <a><img src="exercise.jpg"></a> would be there. But <em>do the
                exercise anyway!</em>
        </div>
    </div>
</body>
</html>
```

HTML structure

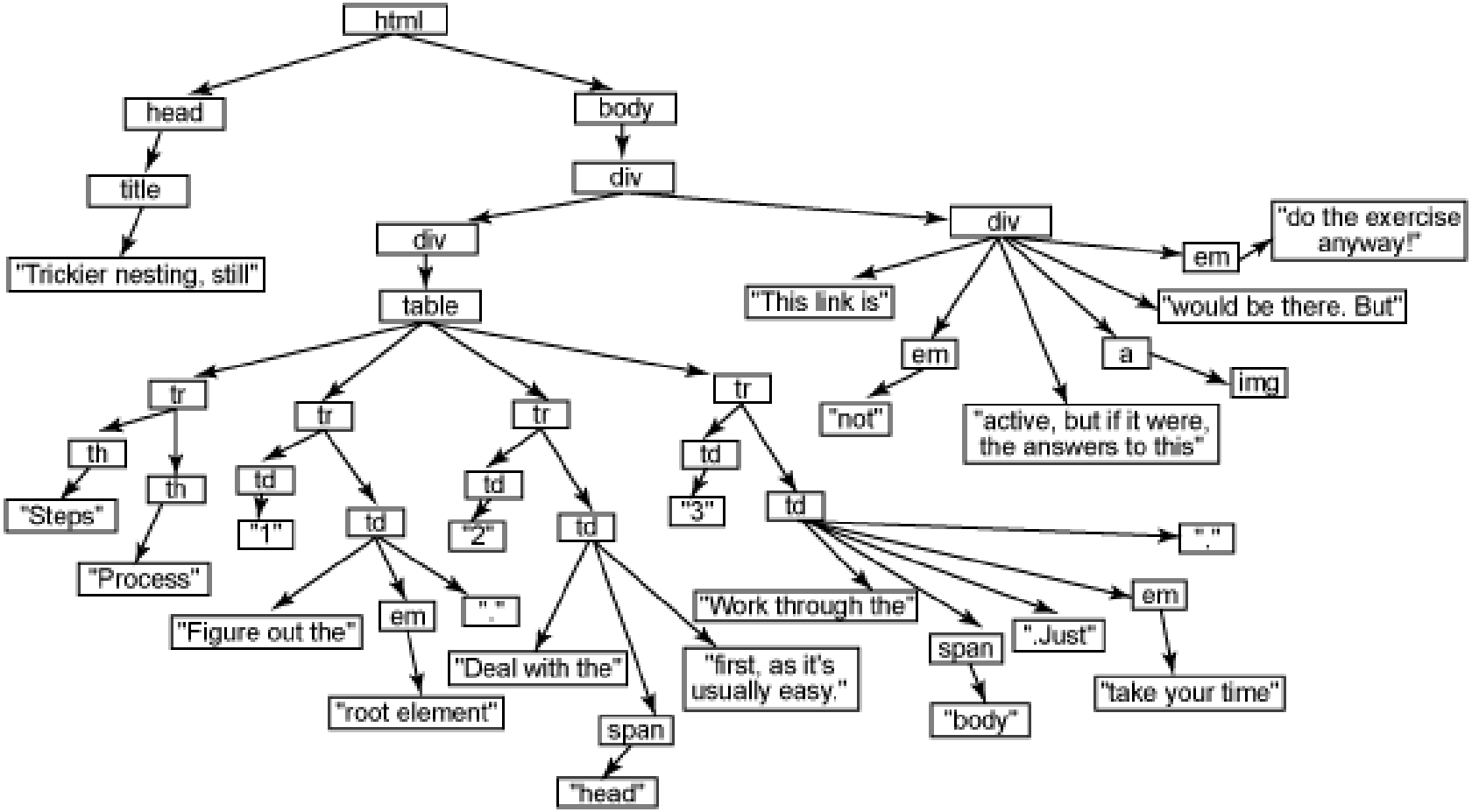
```
<html lang="en">
<head>
    <title>Trickier nesting, still</title>
</head>
<body>
    <div>
        <div>
            <tabl
                <
                <
                <
                <
            </tab
        </div>
        <div>
            This link is <em>not</em> active, but it it were, the answer to this
            <a><img src="exercise.jpg"></a> would be there. But <em>do the
                exercise anyway!</em>
        </div>
    </div>
</body>
</html>
```

Steps	Processes
1	Figure out the *root element*.
2	Deal with the head first as it's usually easy.
3	Work through the body. Just *take your time*.

This link is *not* active, but it it were, the answer to this  would be there. But *do the exercise anyway!*

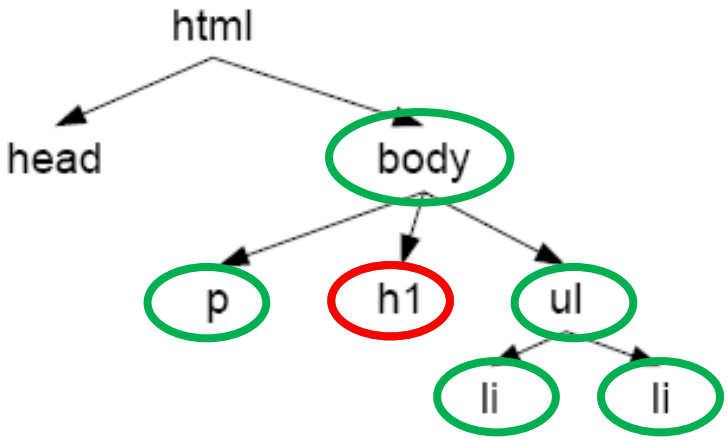
HTML structure

- HTML documents are trees



Tree structure and inheritance

- XHTML documents are trees
- Styles are inherited along trees



- When two rules are in conflict the most specific wins
- Example
  - body {color: green}
  - h1 {color: red}

Example

```
<style type="text/css">
  th { color:green; font-size: 24pt;}
</style>
```

Steps	Processes
1	Figure out the *root element*.
2	Deal with the head first as it's usually easy.
3	Work through the body. Just *take your time*.

This link is *not* active, but it it were, the answer to this  would be there. But *do the exercise anyway!*

```
This link is <em style="color:red; ">not</em> active, …
```

Example

Steps	Processes
1	Figure out the *root element*.
2	Deal with the **head** first as it's usually easy.
3	Work through the **body**. Just *take your time*.

This link is *not* active, but it it were, the answer to this  would be there. But *do the exercise anyway!*

```
<style type="text/css">
  th { color:green; font-size: 24pt; }
  td { text-align: center; }
  span { font-weight: bold; }
</style>
```

Example

Steps	Processes
1	Figure out the *root element*.
2	Deal with the **head** first as it's usually easy.
3	Work through the **body**. Just *take your time*.

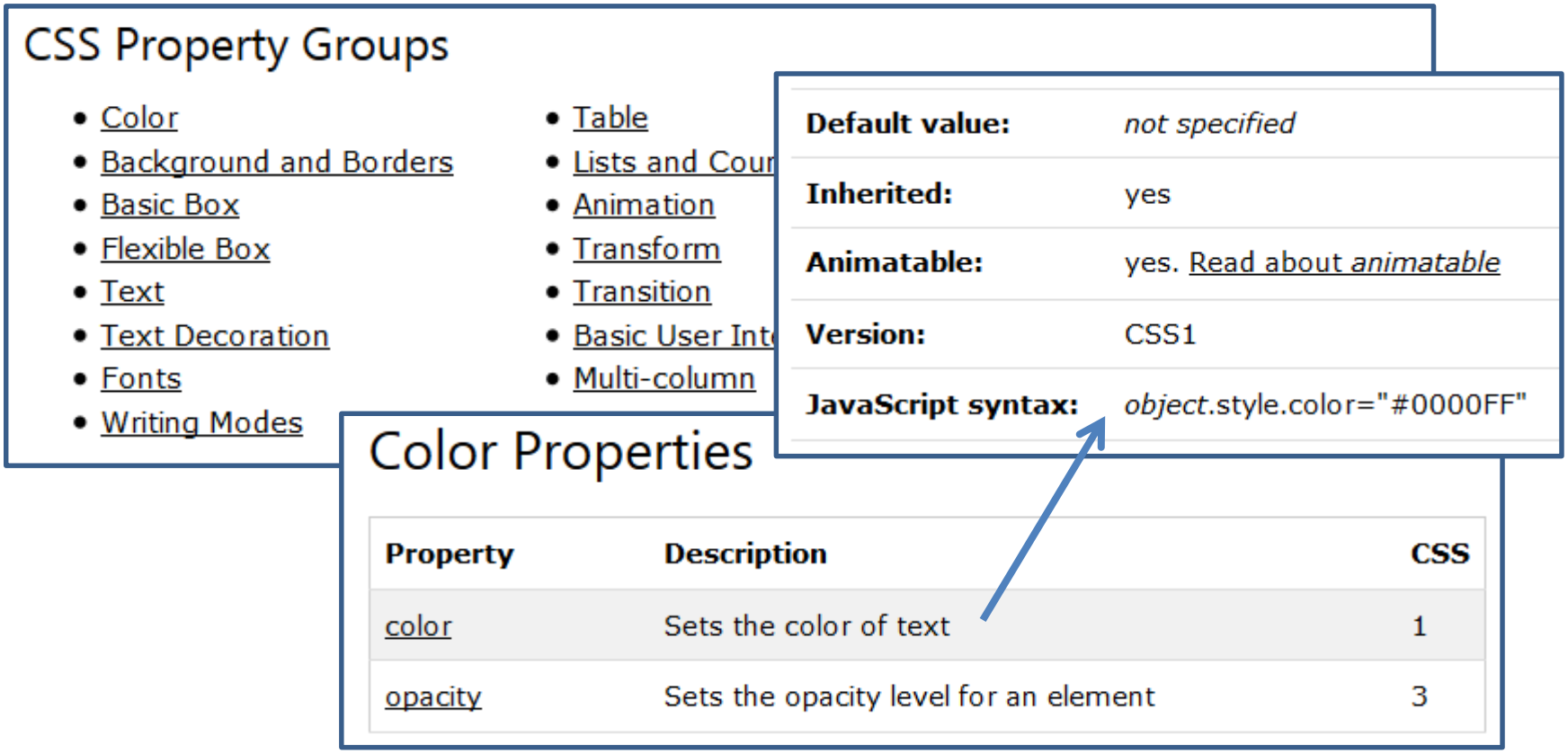
This link is *not* active, but it it were, the answer to this  would be there. But *do the exercise anyway!*

does not change

```
<style type="text/css">
  th { color:green; font-size: 24pt; }
  td { text-align: center; }
  span { font-weight: bold; }
  em { color:brown; }
</style>
```

CSS properties

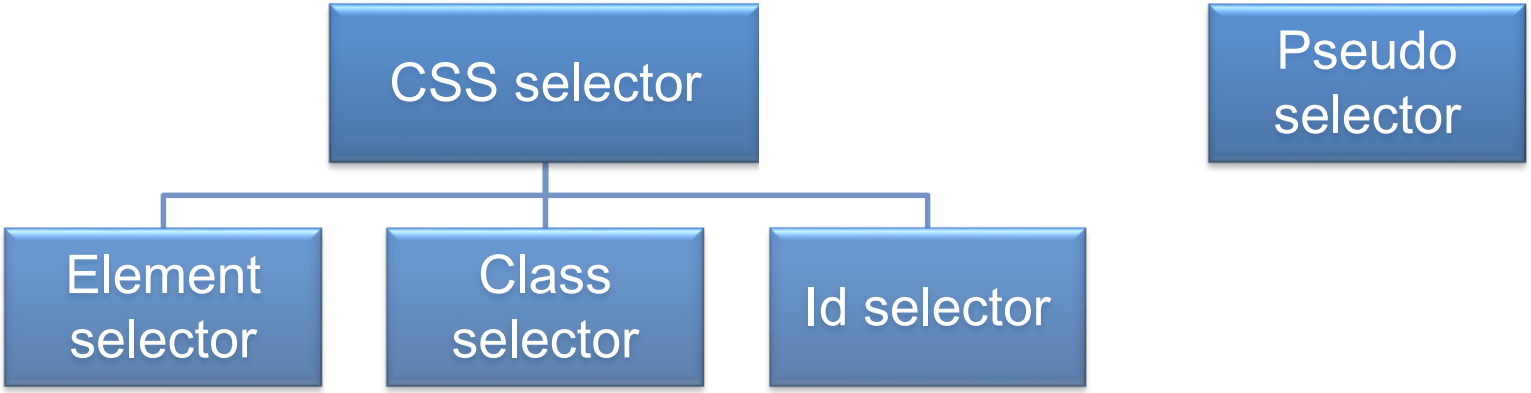
- http://www.w3schools.com/cssref/



CSS SELECTORS

CSS selectors

- Patterns used to select the element(s) you want to style
- Three types of selectors plus “pseudo-selectors”



Element selector

- Used to apply the same style to all instance of a specific element in a document
- In an element selector you choose an element with its tag name and apply the style on that
- Example: apply the color red to all h1 elements that appear in the document

stile.css

```
h1
{
     color : red;
}
```

Class selector

- Used to apply the same style to all elements belonging to a specific (defined) class
- Usually you use the class selector when you want to apply a specific style to a different set of elements
- Example: apply the color blue style to various elements on the document
  - Declare a class name for all elements that must be blue and then select them on the basis of that class name

doc.html

```
<h1 id="titlemessage">Education is must</h1>
    <p class="bluetext">Education in its general sense is a
       form of learning in which […]
    </p>
<h3 class="bluetext">Try to teach</h3>
```

stile.css

```
.bluetext
{
    color:blue;
}
```

Id selector

- Used to apply a style to a specific element in a document
- You can select an element by its (declared) id and apply a style to that
- Example: apply the color grey to the “titlemessage” h1 element

doc.html

```
<h1 id="titlemessage">Education is must</h1>
    <p class="bluetext">Education in its general sense is a
       form of learning in which […]
    </p>
<h3 class="bluetext">Try to teach</h3>
```

stile.css

```
#titlemessage
{
     color : gray
}
```

Pseudo class selector

- Used to style an element based on something other than the structure of the document
  - E.g., the status of a form element or link

```
/* makes all unvisited links blue */
a:link {color:blue;}
/* makes all visited links green */
a:visited {color:green;}
/* makes links red when hovered or activated */
a:hover, a:active {color:red;}
/* makes table rows red when hovered over */
tr:hover {background-color: red;}
/* makes input elements yellow when focus is applied */
input:focus {background-color:yellow;}
```

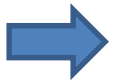
Meaningful HTML

- Meaningful elements
  - h1, h2, ...
  - ul, ol, and dl
  - strong and em
  - blockquote and cite
  - abbr, acronym, and code
  - fieldset, legend, and label
  - caption, thead, tbody, and tfoot
- Id and class names
  - Allow to give extra meaning
- Div and span
  - Add structure to document

Div element

- Stands for “division”
- Used to group block-level elements
  - Provides a way of dividing a document into meaningful areas
- Use only if necessary and not redundant

```
<div id="mainNav">
  <ul>
    <li>Home</li>
    <li>About Us</li>
    <li>Contact</li>
  </ul>
</div>
```



```
<ul id="mainNav">
  <li>Home</li>
  <li>About Us</li>
  <li>Contact</li>
</ul>
```

Span element

- Used to group or identify inline elements

```
<h2>Where's Durstan?</h2>
<p>Published on
     <span class="date">March 22nd, 2005</span>
by <span class="author">Andy Budd</span></p>
```

CSS selectors

Selector	Example	Example description	CSS
.class	.intro	Selects all elements with class="intro"	1
#id	#firstname	Selects the element with id="firstname"	1
*	*	Selects all elements	2
element	p	Selects all <p> elements	1
element,element	div, p	Selects all elements and all <p> elements	1
element element	div p	Selects all <p> elements inside elements	1
element>element	div > p	Selects all <p> elements where the parent is a element	2
element+element	div + p	Selects all <p> elements that are placed immediately after elements	2
element1~element2	p ~ ul	Selects every element that are preceded by a <p> element	3

http://www.w3schools.com/cssref/css_selectors.asp

CSS selectors

Selector	Example	Example description	CSS
[*attribute*]	[target]	Selects all elements with a target attribute	2
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"	2
[*attribute~=value*]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"	2
[*attribute|=value*]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"	2
[*attribute^=value*]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"	3
[*attribute$=value*]	a[href$=".pdf"]	Selects every <a> element whose href attribute value ends with ".pdf"	3
[*attribute*=value*]	a[href*="w3schools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"	3

CSS selectors

Selector	Example	Example description	CSS
:active	a:active	Selects the active link	1
::after	p::after	Insert something after the content of each <p> element	2
::before	p::before	Insert something before the content of each <p> element	2
:checked	input:checked	Selects every checked <input> element	3
:disabled	input:disabled	Selects every disabled <input> element	3
:empty	p:empty	Selects every <p> element that has no children (including text nodes)	3
:enabled	input:enabled	Selects every enabled <input> element	3
:first-child	p:first-child	Selects every <p> element that is the first child of its parent	2
::first-letter	p::first-letter	Selects the first letter of every <p> element	1
::first-line	p::first-line	Selects the first line of every <p> element	1

CSS selectors

Selector	Example	Example description	CSS
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent	3
:focus	input:focus	Selects the input element which has focus	2
:hover	a:hover	Selects links on mouse over	1
:in-range	input:in-range	Selects input elements with a value within a specified range	3
:invalid	input:invalid	Selects all input elements with an invalid value	3
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute equal to "it" (Italian)	2
:last-child	p:last-child	Selects every <p> element that is the last child of its parent	3
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent	3
:link	a:link	Selects all unvisited links	1

CSS selectors

Selector	Example	Example description	CSS
:not(*selector*)	:not(p)	Selects every element that is not a <p> element	3
:nth-child(*n*)	p:nth-child(2)	Selects every <p> element that is the second child of its parent	3
:nth-last-child(*n*)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child	3
:nth-last-of-type(*n*)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child	3
:nth-of-type(*n*)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent	3
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent	3
:only-child	p:only-child	Selects every <p> element that is the only child of its parent	3
:optional	input:optional	Selects input elements with no "required" attribute	3
:out-of-range	input:out-of-range	Selects input elements with a value outside a specified range	3

CSS selectors

Selector	Example	Example description	CSS
:read-only	input:read-only	Selects input elements with the "readonly" attribute specified	3
:read-write	input:read-write	Selects input elements with the "readonly" attribute NOT specified	3
:required	input:required	Selects input elements with the "required" attribute specified	3
:root	:root	Selects the document's root element	3
::selection	::selection	Selects the portion of an element that is selected by a user	
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)	3
:valid	input:valid	Selects all input elements with a valid value	3
:visited	a:visited	Selects all visited links	1

Selectors

- Examples of new CSS3 pseudo-classes

```
:nth-child(3n)
```


```
:nth-child(3n+2)
```


```
:nth-child(-n+4)
```

Selectors

- N = pseudoclass expressions

n	2n+1	4n+1	4n+4	4n	5n-2	-n+3
0	1	1	4	-	-	3
1	3	5	8	4	3	2
2	5	9	12	8	8	1
3	7	13	16	12	13	-
4	9	17	20	16	18	-
5	11	21	24	20	23	-

Selectors

- Examples of new CSS3 pseudo-classes

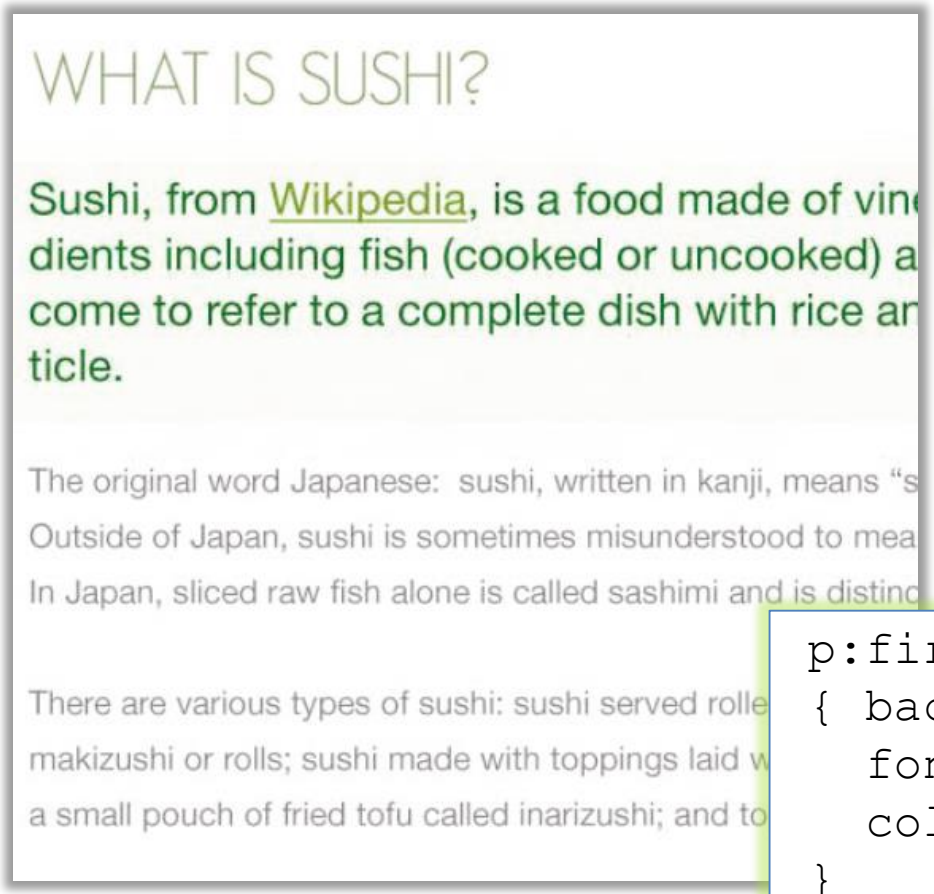
odd = 2n+1
even = 2n


```
tr:nth-child(odd) td
{ background: #ecffd9; }
```


```
tr:nth-child(-n+3) td
{ background: #ecffd9; }
```

Selectors

- Examples of new CSS3 pseudo-classes




```
:not (:first-child)
{ font-size: 75%; }
```

```
p:first-of-type
{ background: #fafcf5;
  font-size: 1.3em;
  color: #030;
}
```

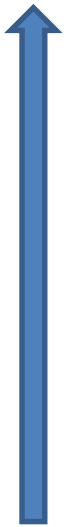
Selectors

- Pseudo-elements
  - ::selection
    Targets elements that have been highlighted by the user
- Example: match any user-selected text within a textarea element

```
textarea::selection
```

Cascading Style Sheets

- The term “cascading” means that a document can include more than one style sheet
- In this case, visualization follows priority rules
  - User Style
  - Inline Style (inside HTML tag)
  - Internal Style (usually in the HTML head section)
  - External Style
  - Browser Default Style



External style

- Link to an external style sheet using the <link> element

stile.css

```
h1 { font-size:17px;
   font-family:verdana; color:green; }
h2 { font-size:18px;
   font-family:arial; color:red; }
```

```
<head>
<link rel=stylesheet type="text/css"
      href="stile.css">
</head>
<body>
<h1>Questo testo e' di colore verde, e utilizza il
font verdana a 17 pixel</h1>
<h2>Questo testo e' di colore rosso, e utilizza il
font arial a 18 pixel</h2>
</body>
```

External style

- Alternative method
- @import directive in the <style> element

```
<head>
  <style>
    @import url(stile.css);
  </style>
</head>
<body>
...
</body>
```

Internal style

- <style> element inside the document header

```
<head>
<style type="text/css">
h1 { font-size:17px; font-family:verdana;
color:green; }
h2 { font-size:18px; font-family:arial;
color:red; }
</style>
</head>
```

Inline style

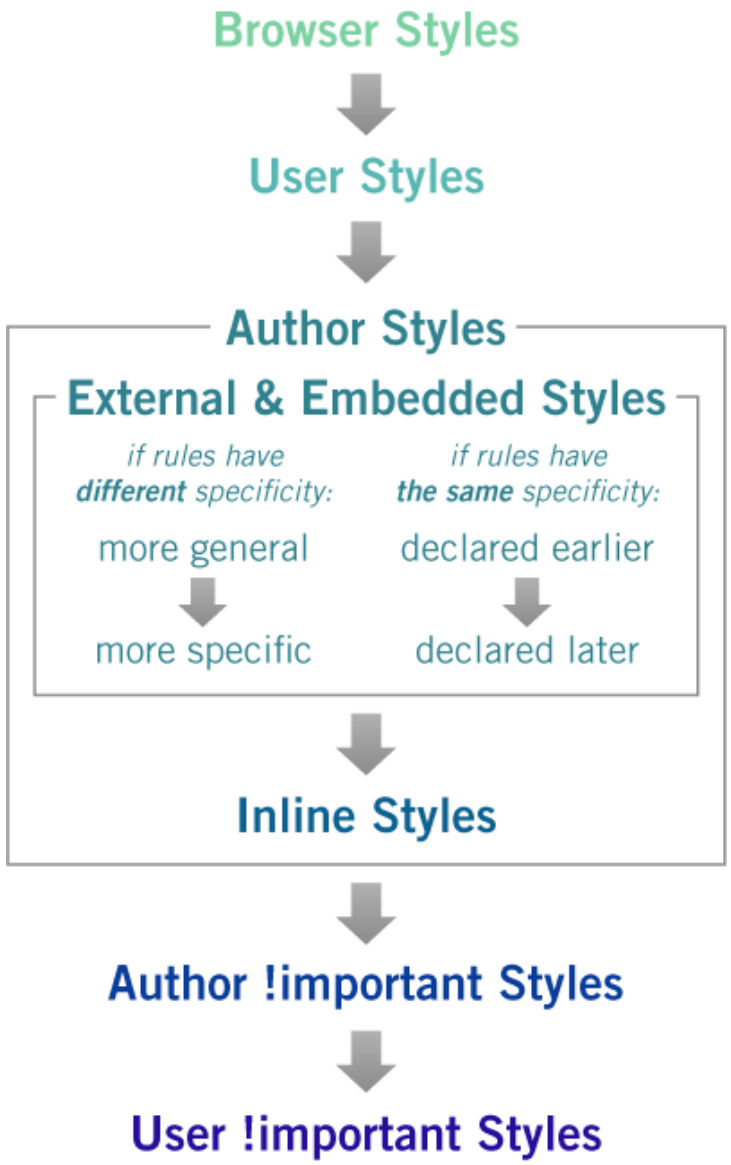
- <style> attribute within an HTML element

```
<h1 style="font-size:17px;
  font-family:verdana; color:green; "> Questo
  testo e' di colore verde, e utilizza il
  font  verdana a 17 pixel </h1>
```

Priority rules

- Rules can be marked as “important”

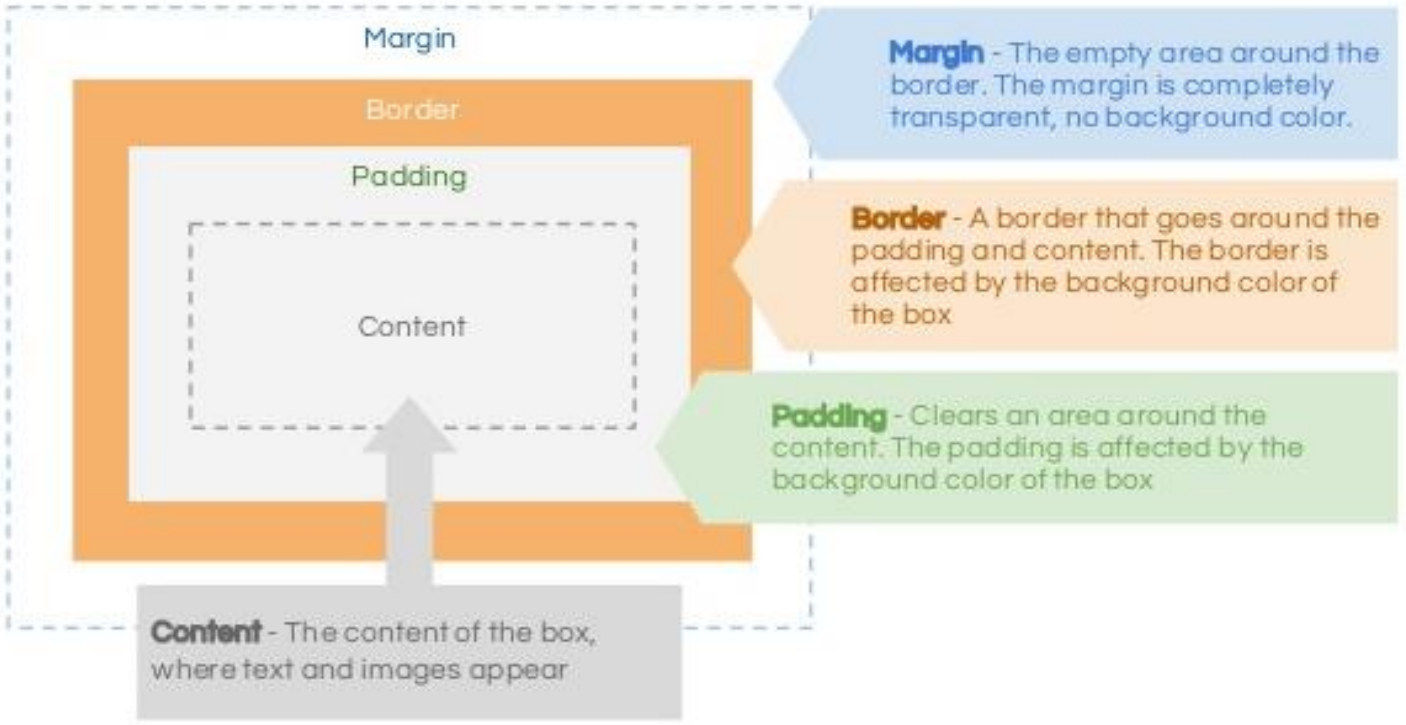
```
h1 {
  color:red !important
}
```



CSS BOX MODEL

The box model

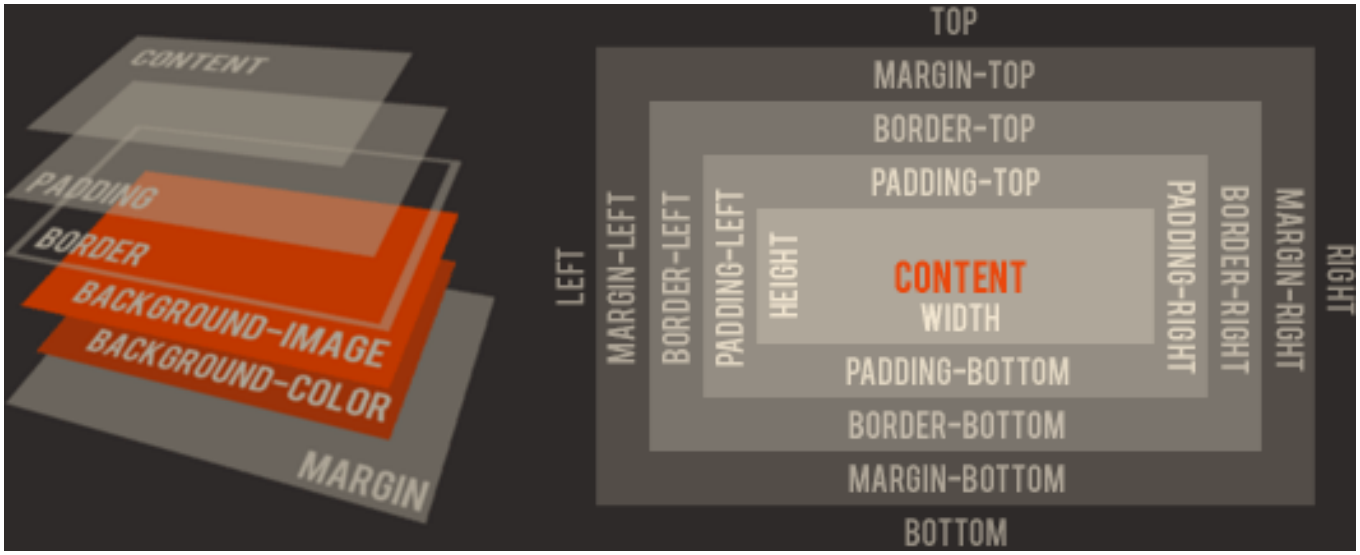
- One of the cornerstones of CSS
- Every element on the page is considered to be a rectangular box



The box model

http://www.w3schools.com/Css/css_boxmodel.asp

- Total element width = width + left padding + right padding + left border + right border + left margin + right margin
- Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

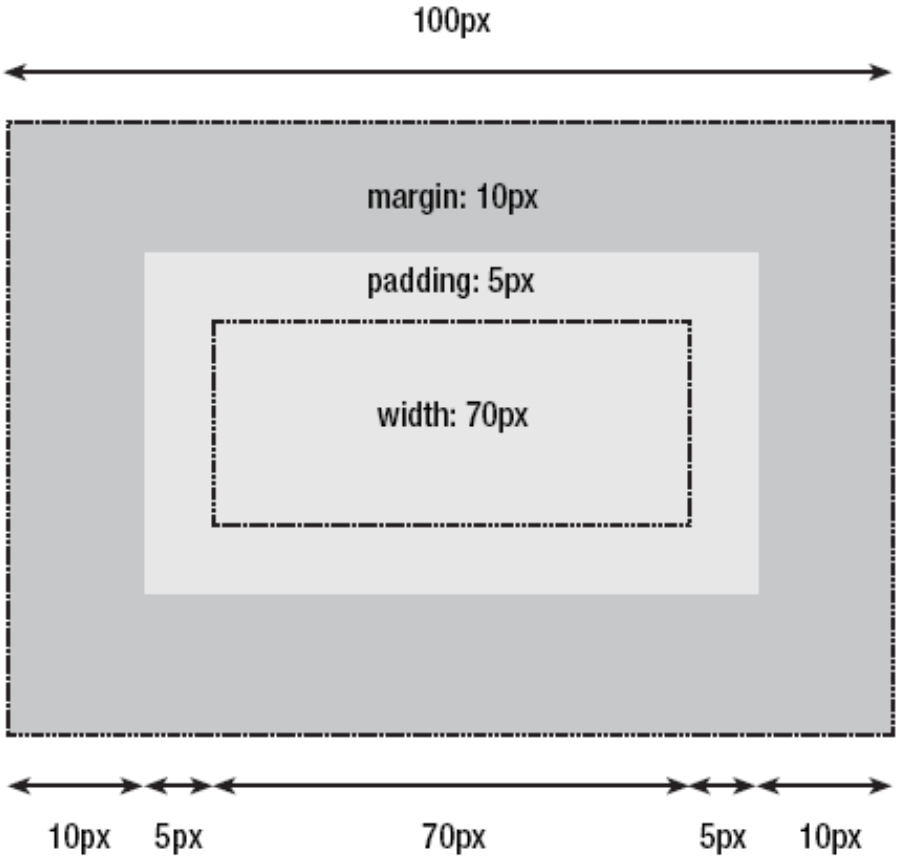


- You can set any of these properties, independently

Example

- Padding, borders, and margins are optional and default to zero

```
#myBox {
  margin: 10px;
  padding: 5px;
  width: 70px;
}
```



Positioning schemes

- In CSS there are three basic positioning schemes
  - Normal flow
  - Floats
  - Absolute positioning
- Unless specified, all boxes start life being positioned in the normal flow
  - The position of an element's box in the normal flow is dictated by that element's position in the HTML code

Normal flow

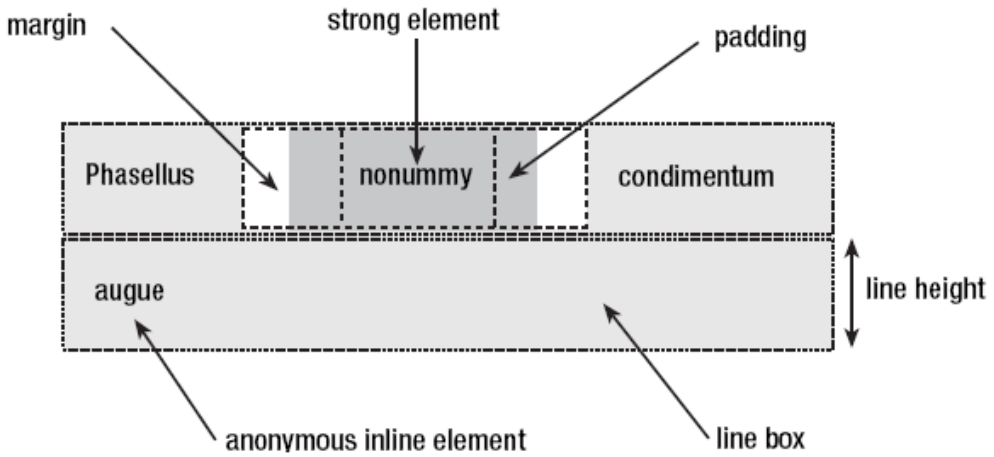
- Block-level boxes will appear vertically one after the other

```
<div> … </div>
```

  - The vertical distance between boxes is calculated by the boxes' vertical margins
- Inline boxes are laid out in a line horizontally

```
<span> … </span>
```

  - Their horizontal spacing can be adjusted using horizontal padding, borders, and margins
  - Vertical padding, borders, and margins will have no effect on the height of an inline box



Display property

- Allows to control element visualization (block or inline)
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way

```
li {display:inline;}
```

```
span {display:block;}
```

- Example

http://www.w3schools.com/Css/css_display_visibility.asp

Display and visibility properties

- The property display allows to hide an element, too
  - The element will be hidden, and the page will be displayed as if the element is not there

```
h1.hidden {
    display: none;
}
```

- The property visibility also can hide an element, but the element will still take up the same space as before
  - The element will be hidden, but still affects the layout

```
h1.hidden {
    visibility: hidden;
}
```

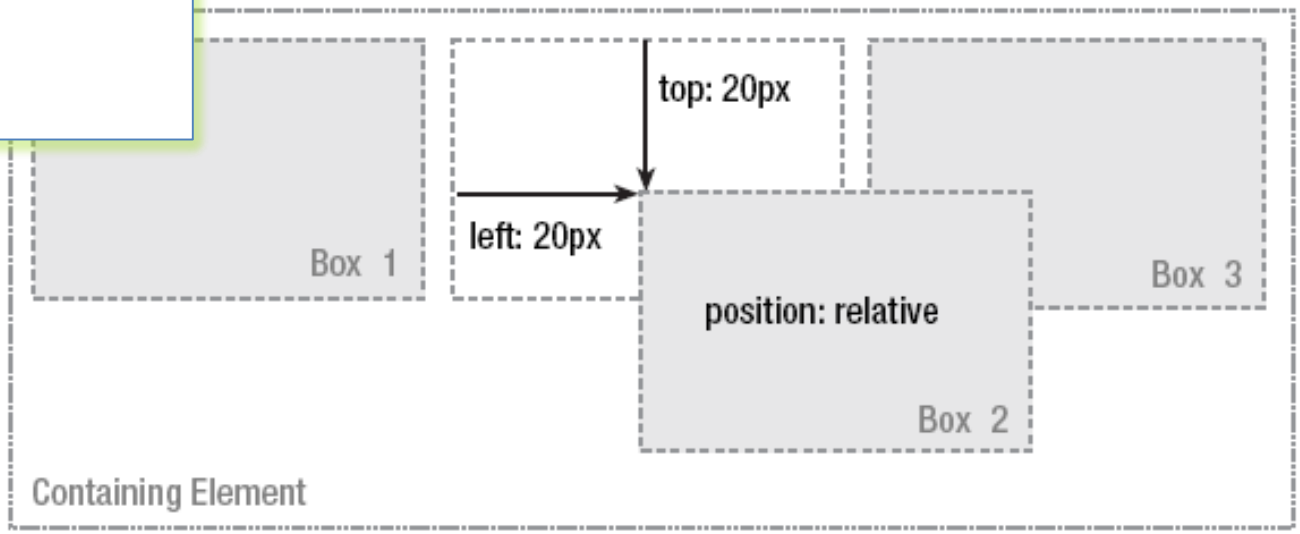
Box positioning

- A block can be positioned in different ways to which correspond different positioning schemes
  - Static: normal flow
  - Relative: the offset values are relative to the block position in the normal flow
  - Absolute: the box position is determined by the top, left, right, bottom properties and is relative to the containing block
  - Fixed: the box is fixed with respect to some reference point (the viewport as an example)

Relative positioning

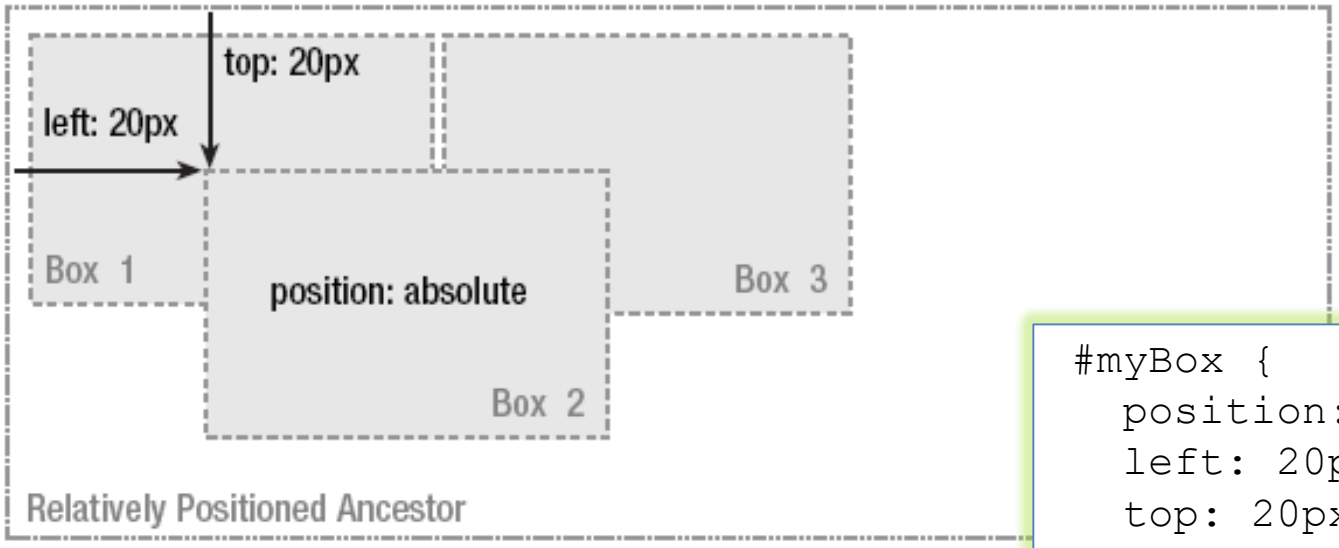
- It is possible to shift one element “relative” to its starting point by setting a vertical or horizontal offset

```
#myBox {
  position: relative;
  left: 20px;
  top: 20px;
}
```



Absolute positioning

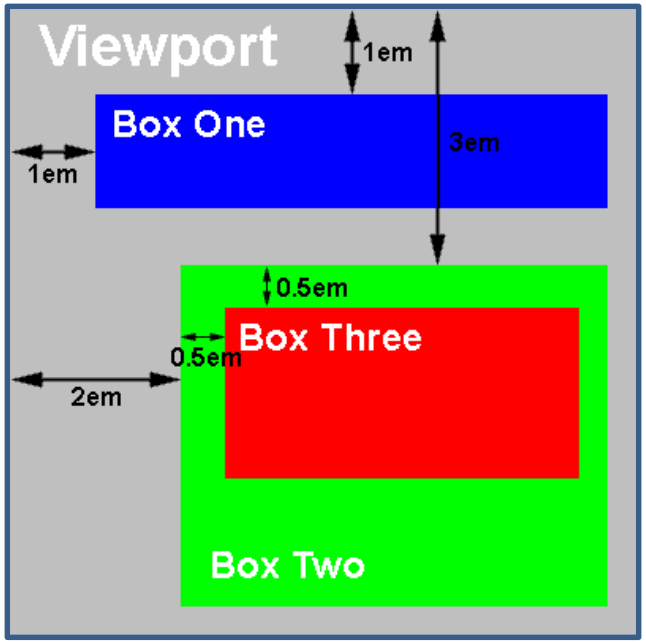
- Takes the element out of the flow of the document, thus taking up no space
- Other elements in the normal flow of the document will act as though the absolutely positioned element was never there

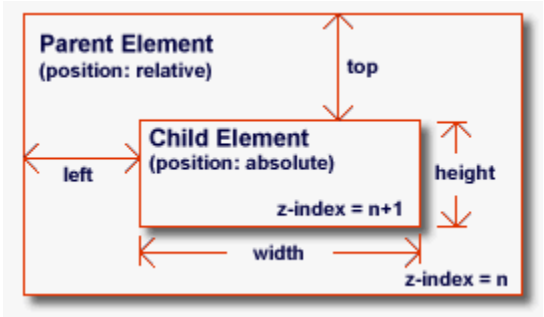



```
#myBox {
  position: absolute;
  left: 20px;
  top: 20px;
}
```

Fixed positioning

- A subcategory of absolute positioning
  - The container block is the viewport
- Allows to create elements that always stay at the same position in the window
- In case of overlaps the z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others)

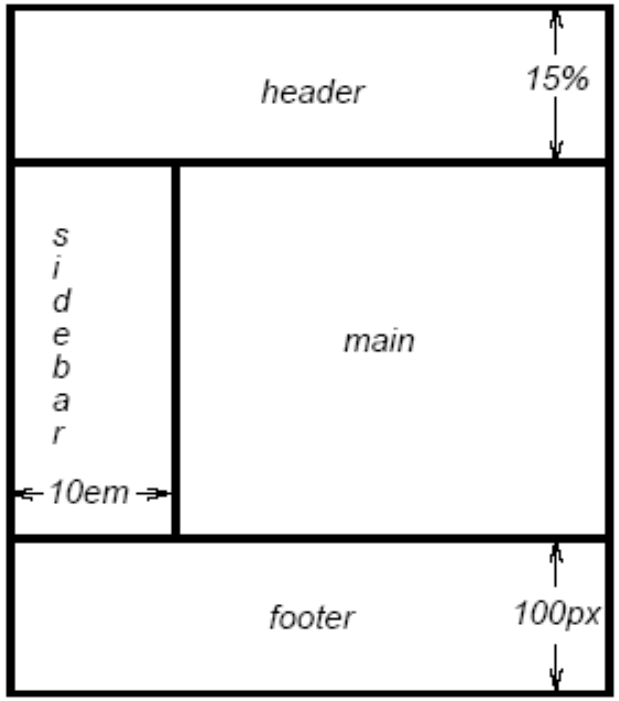




Fixed positioning

- Can be used to create complex frame-like presentations

```
#header { position: fixed; width: 100%;
height: 15%; top: 0; right: 0;
bottom: auto; left: 0; }
#sidebar { position: fixed; width: 10em;
height: auto; top: 15%; right: auto;
bottom: 100px; left: 0;}
#main {position: fixed; width: auto;
height: auto; top: 15%; right: 0; bottom: 100px;
left: 10em; }
#footer {position: fixed; width: 100%;
height: 100px; top: auto; right: 0;
bottom: 0; left: 0; }
```

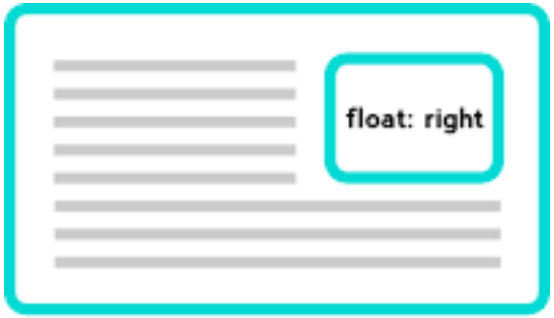


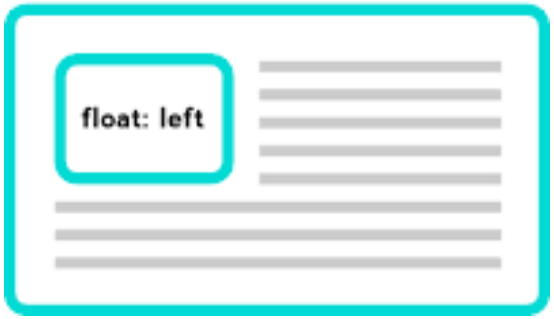
http://www.w3schools.com/Css/css_positioning.asp

Floating

- A floated box can either be shifted to the left or the right until its outer edge touches the edge of its containing box, or another floated box
- Often used for images and when working with layouts

```
img
{
float:right;
}
```

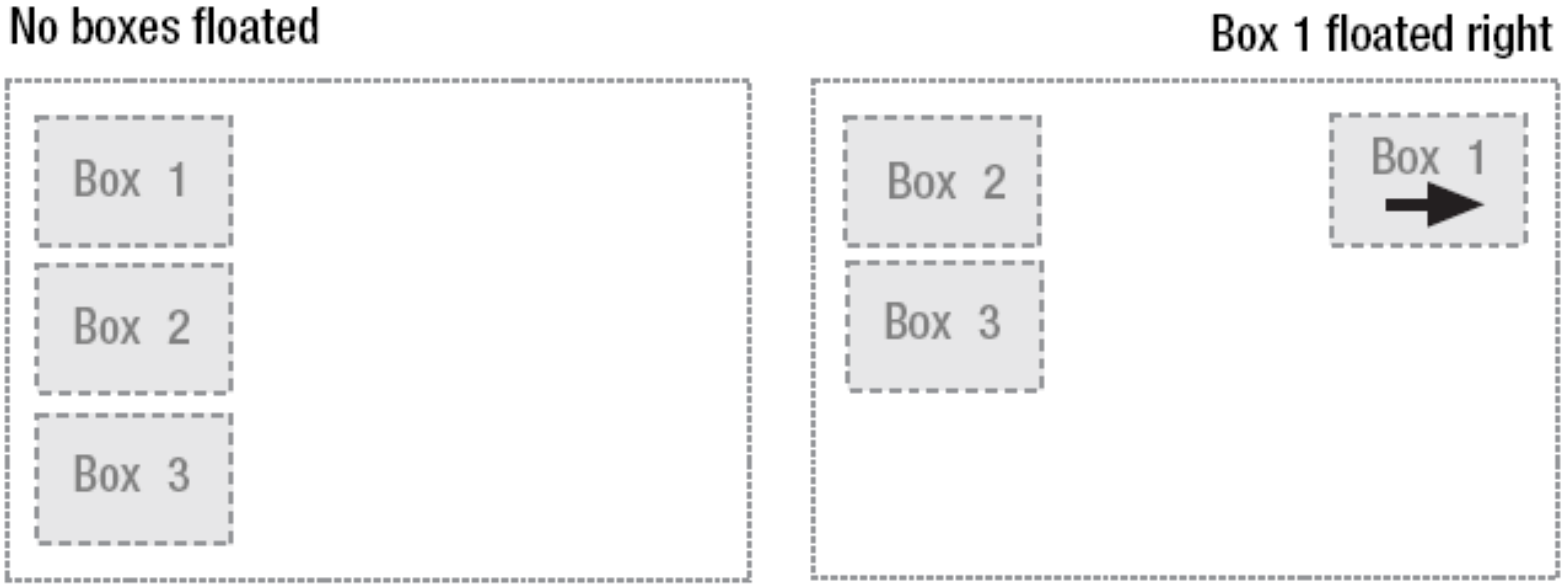




http://www.w3schools.com/Css/css_float.asp

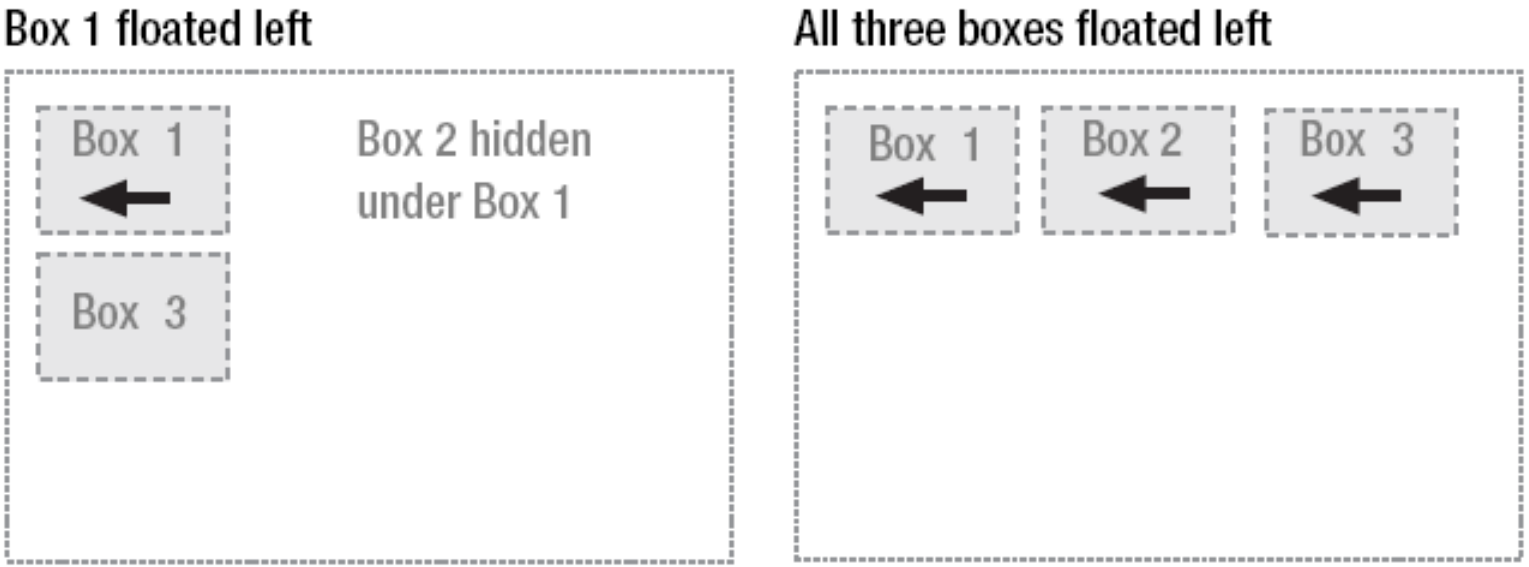
Floating

- Floated boxes aren't in the normal flow of the document, so block boxes in the regular flow of the document behave as if the floated box wasn't there



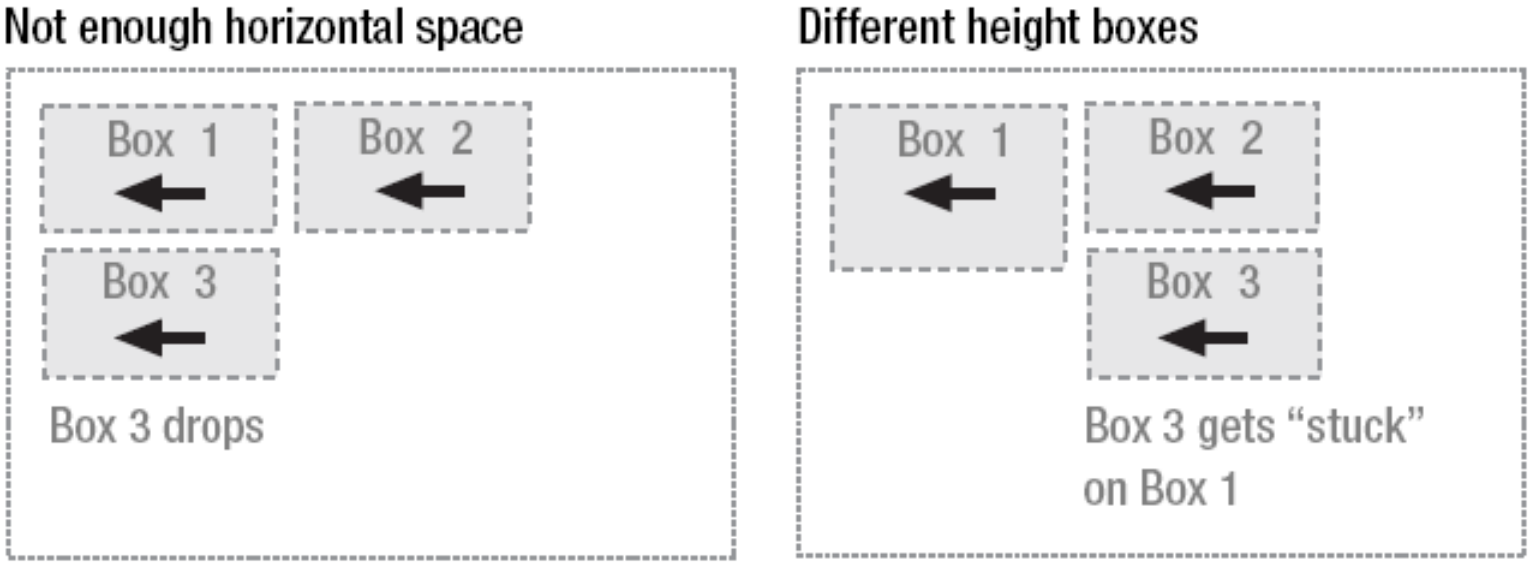
Floating

- If all three boxes are floated left
  - Box 1 is shifted left until it touches its containing box
  - Other two boxes are shifted left until they touch the preceding floated box



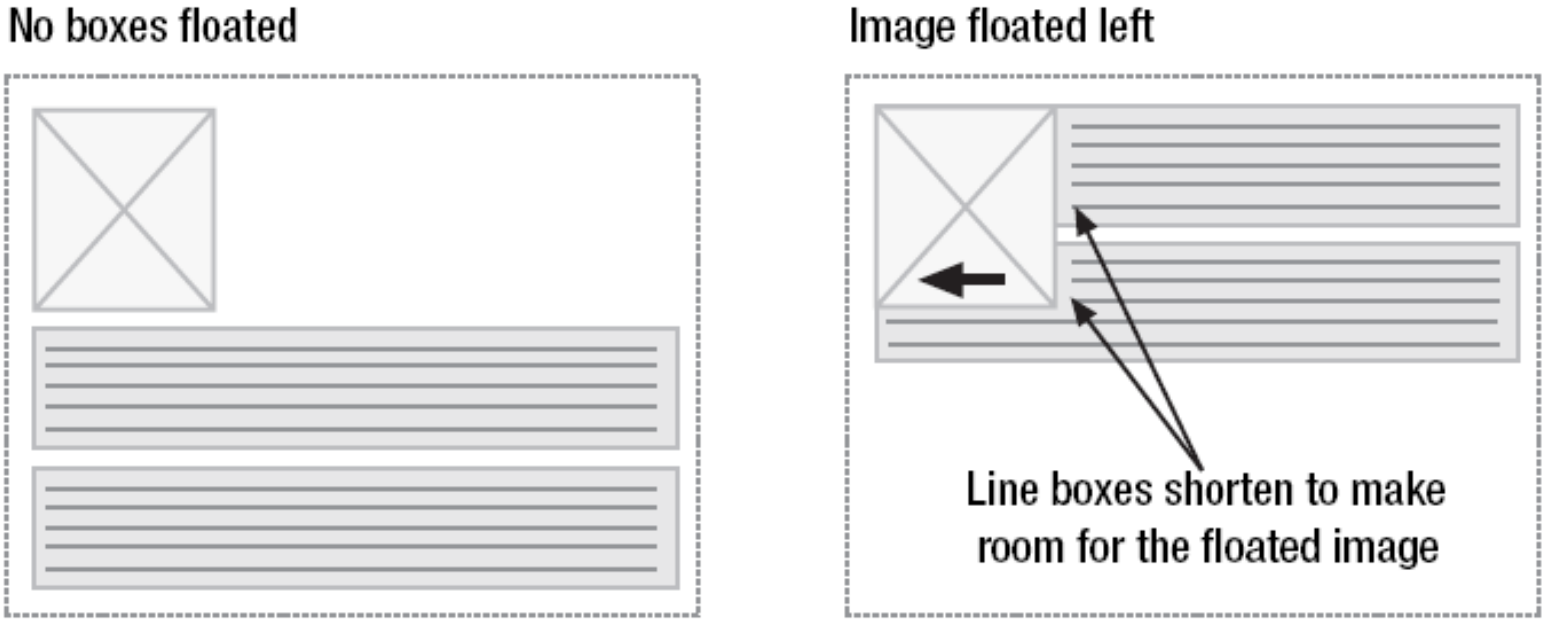
Floating

- If the containing block is too narrow for all of the floated elements to fit horizontally
  - The remaining floats will drop down until there is sufficient space
  - If the floated elements have different heights, it is possible for floats to get “stuck” on other



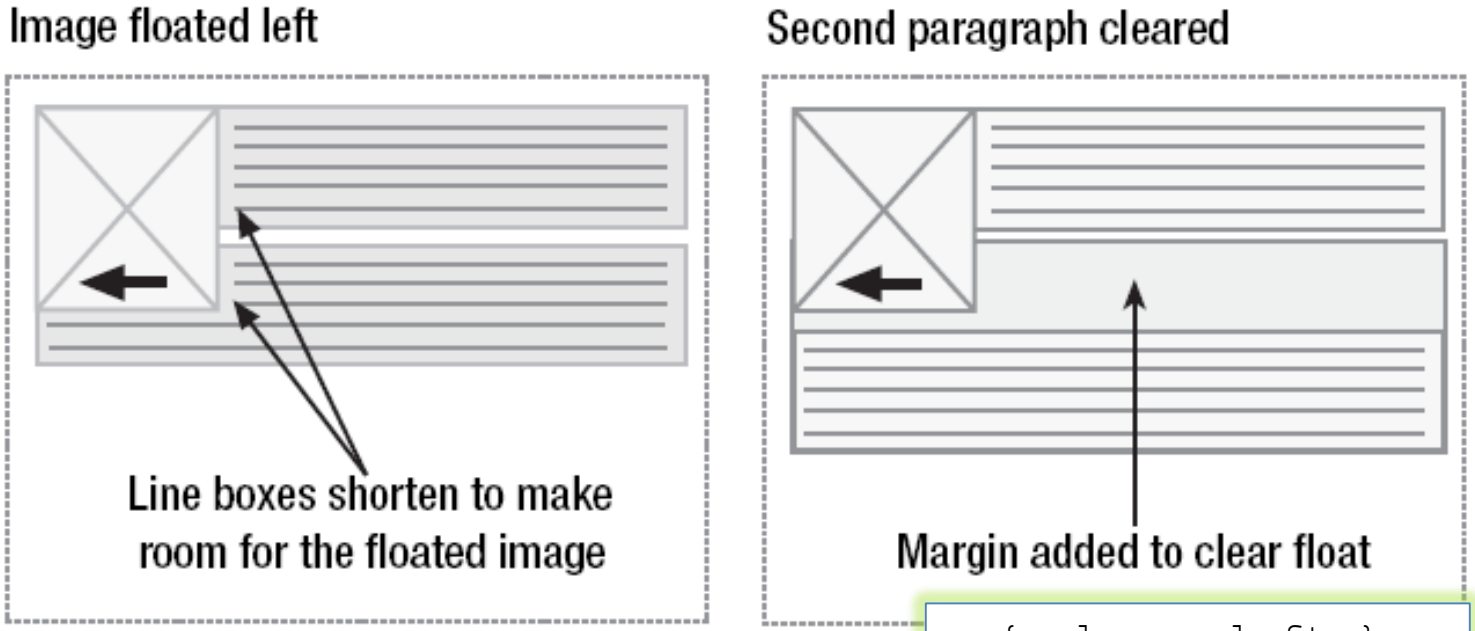
Line boxes and clearing

- Line boxes next to a floated box are shortened to make room for the floated box, and flow around the float
  - Floats were created to allow text to flow around images



Line boxes and clearing

- To stop line boxes flowing around the outside of a floated box, you need to apply a clear to that box
  - The clear property can be left, right, both, or none, and indicates which side of the box should not be next to a floated box




```
p { clear: left }
```

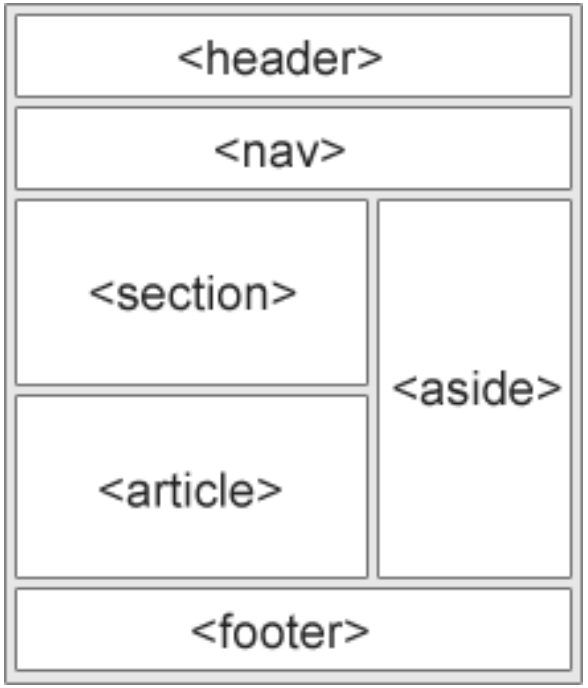
PAGE LAYOUT WITH CSS

Page layout

- Possibility to control page layout without the need to use presentation markup

- Example

HTML5 semantic tags



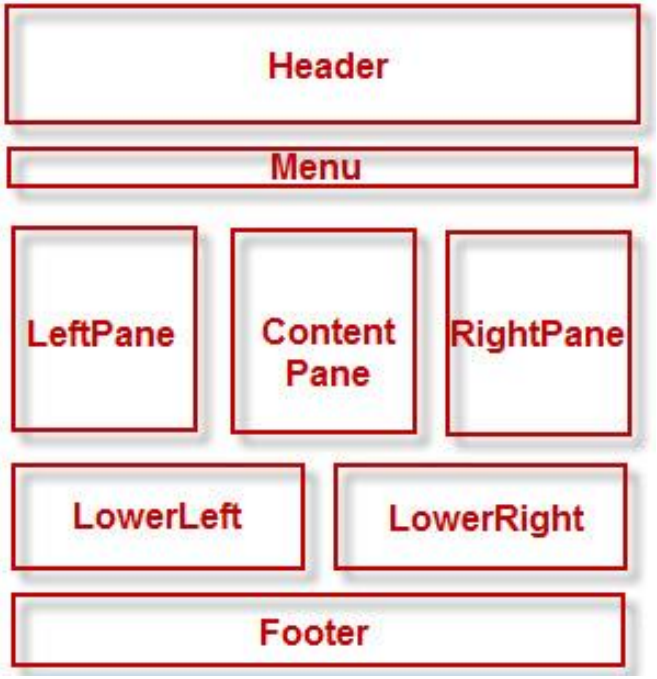
- <header>: defines a header for a document or a section
- <nav>: defines a container for navigation links
- <section>: defines a section in a document
- <article>: defines an independent self-contained article
- <aside>: defines content aside from the content (like a sidebar)
- <footer>: defines a footer for a document or a section
- <details>: defines additional details
- <summary>: defines a heading for the <details> element

Example

```
header {
    background-color:black;
    color:white;
    text-align:center;
    padding:5px; }
nav {
    line-height:30px;
    background-color:#eeeeee;
    height:300px;
    width:100px;
    float:left;
    padding:5px; }
```

```
section {
    width:350px;
    float:left;
    padding:10px; }
footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
    padding:5px; }
```

Example

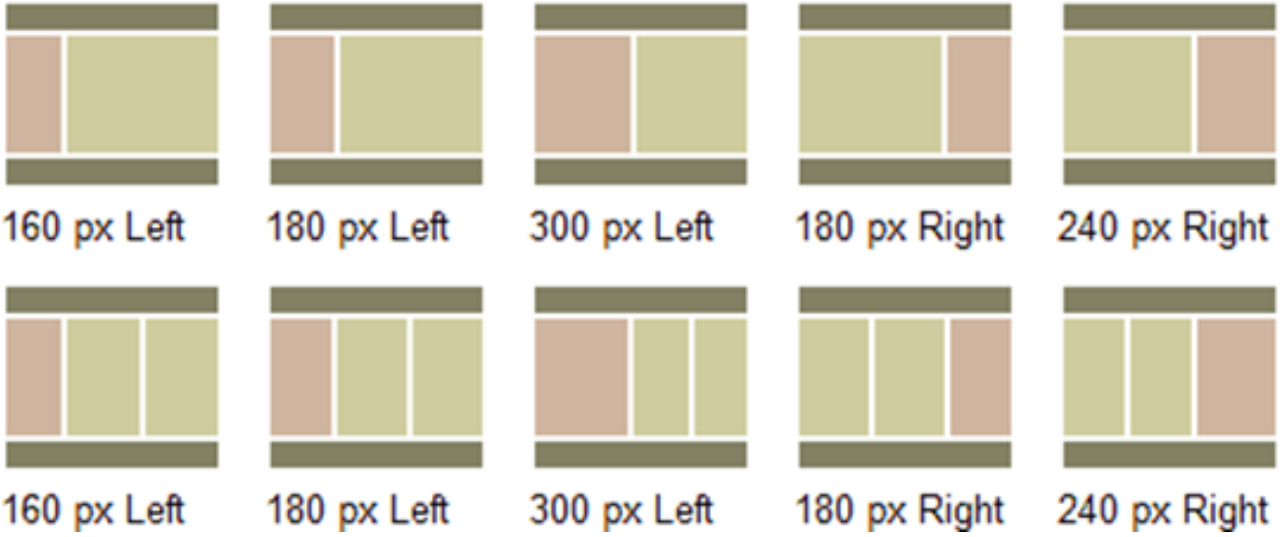



```
<div id="headerWrap">Header</div>
<div id="MenuWrap">Menu</div>
<div id="LeftPane">LeftPane</div>
<div id="ContentPane">ContentPane</div>
<div id="RightPane">RightPane</div>
<div id="LowerLeftPane">LowerLeft</div>
<div id="LowerRightPane">LowerRight</div>
<div id="footerWrap">Footer</div>
```

```
#headerWrap { width:100%;}
#MenuWrap { width:100%;}
#LeftPane { float:left; width:33%;}
#ContentPane { float:left; width:34%;}
#RightPane { float:left; width:33%;}
#LowerLeftPane { clear:both; float:left;
                 width:50%;}
#LowerRightPane { float:left; width:50%;}
#footerWrap { clear:both; width:100%; }
```

Multi-column layout

- How to create an horizontally centered page design
- How to create two- and three-column float-based layouts
- How to create fixed-width, liquid, and elastic layouts



Centering a design

- Long lines of text can be difficult and unpleasant to read
- Rather than spanning the full width of the screen, centered designs span only a portion of the screen, creating shorter and easier-to-read line lengths

- Two basic methods
  - Auto margins
  - Positioning and negative margins

Auto margins

- Define the width of the wrapper div
- Set the horizontal margin to auto

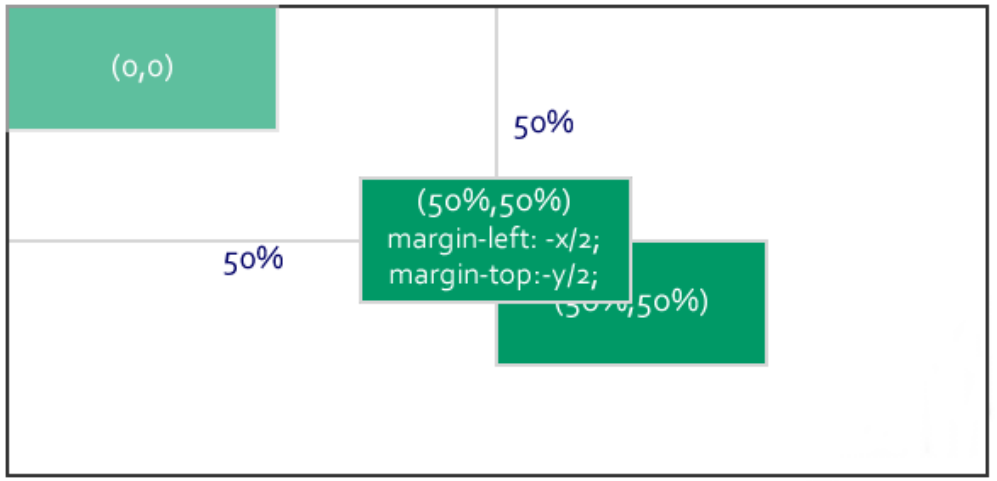
```
<body>
<div id="wrapper">
</div>
</body>
```

```
#wrapper {
width: 200px;
margin: 20px auto;
}
```

Positioning and negative margins

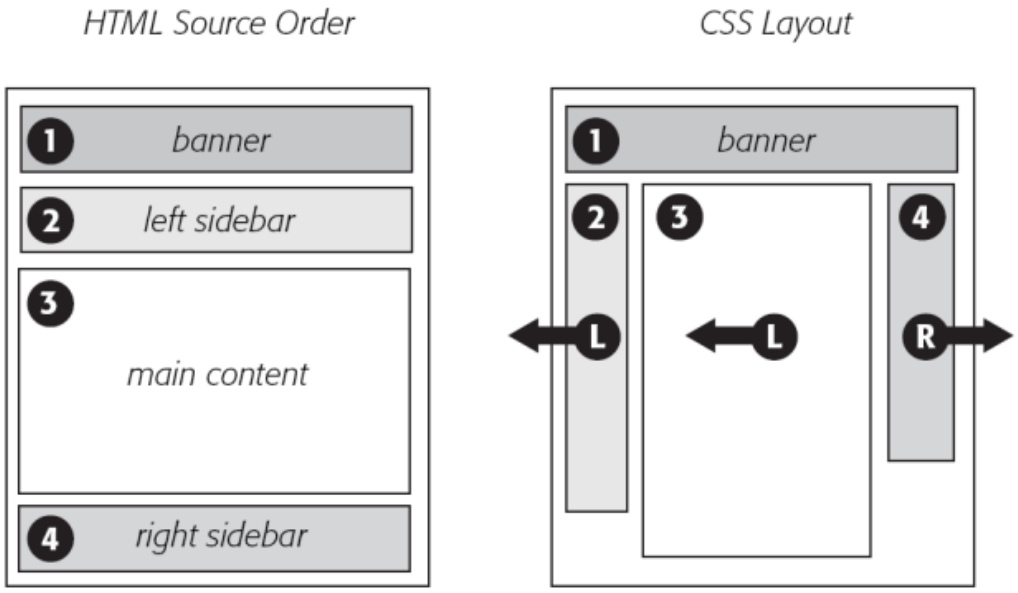
- Define the width of the wrapper div
- Set the position property of the wrapper to relative
- Set the left property to 50%
- Apply a negative margin to the left side of the wrapper, equal to half the width of the wrapper

```
#wrapper {
width: 720px;
position: relative;
left: 50%;
margin-left: -360px;
}
```



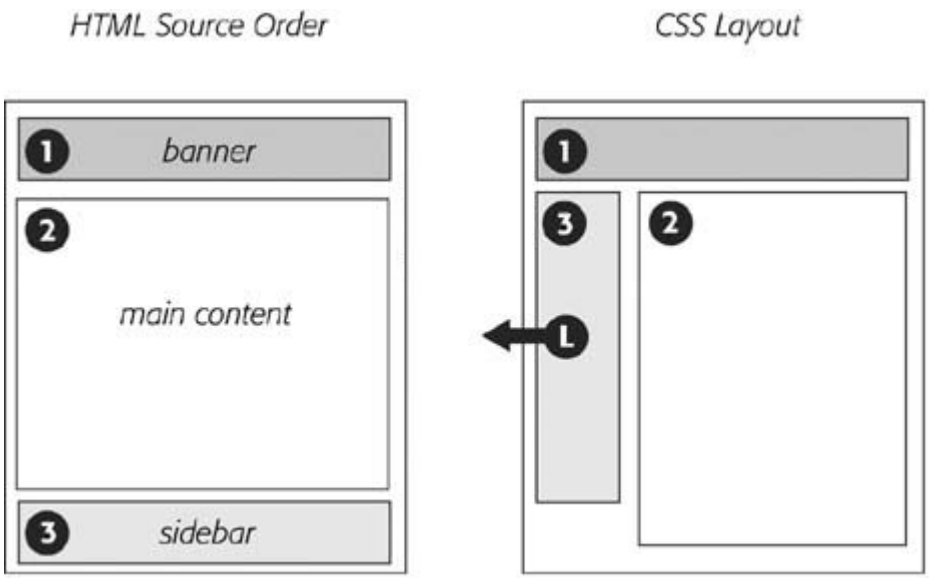
Float-based layouts

- Set the width of the elements you want to position, and then float them left or right
  - Two-column floated layout
  - Three-column floated layout



Two-column floated layout

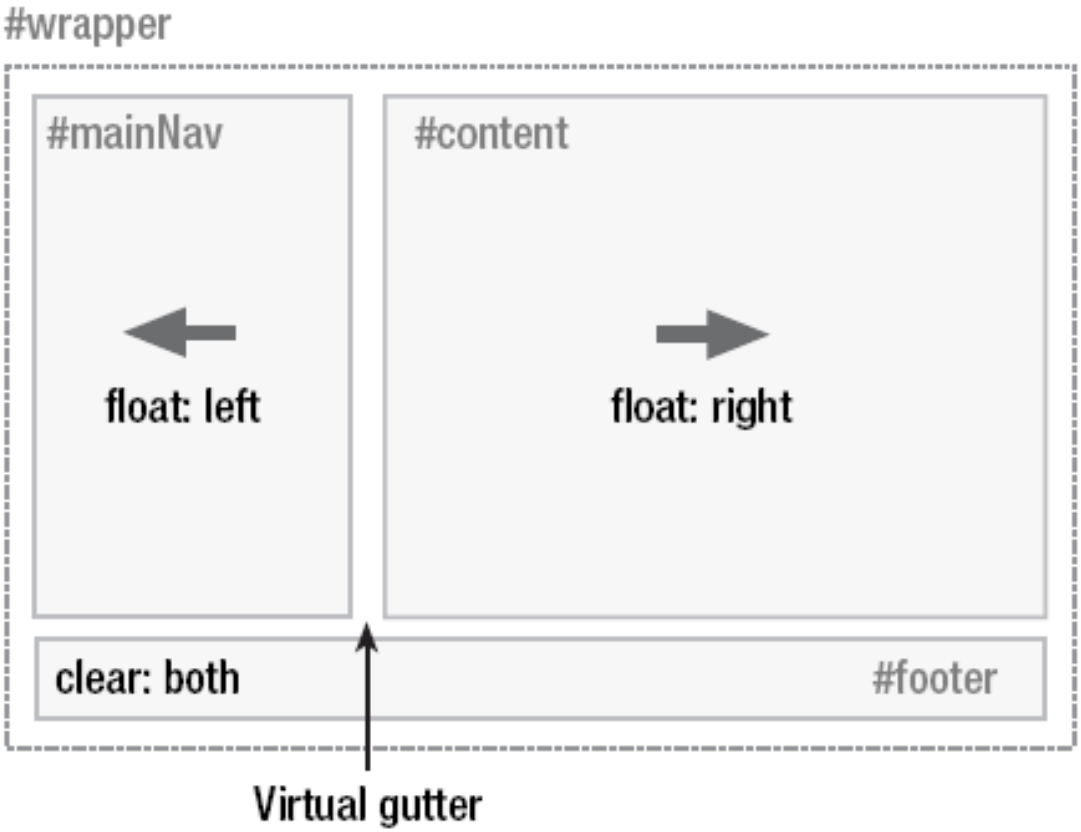
- HTML framework
  - Main navigation on the left side of the page
  - Content on the right
- For accessibility reasons the content area is above the navigation in the source
  - The main content is the most important element in the page and it so should come first in the document
  - There is no point forcing screen-reader users to read through a potentially long list of links before they get to the content



Two-column floated layout

- Create a virtual gutter by floating one element left and one element right

```
<div id="wrapper">
<div id="branding">
...
</div>
<div id="content">
...
</div>
<div id="mainNav">
...
</div>
<div id="footer">
...
</div>
</div>
```



Two-column floated layout

- Better: add horizontal padding

```
#content {
width: 520px;
float: right;
}
#mainNav {
width: 180px;
float: left;
}
#footer {
clear: both;
}
```

```
#mainNav {
padding-top: 20px;
padding-bottom: 20px;
}
#mainNav li {
padding-left: 20px;
padding-right: 20px;
}
#content h1, #content h2,
     #content p {
padding-right: 20px;
}
```

https://blog.html.it/layoutgala/index.html

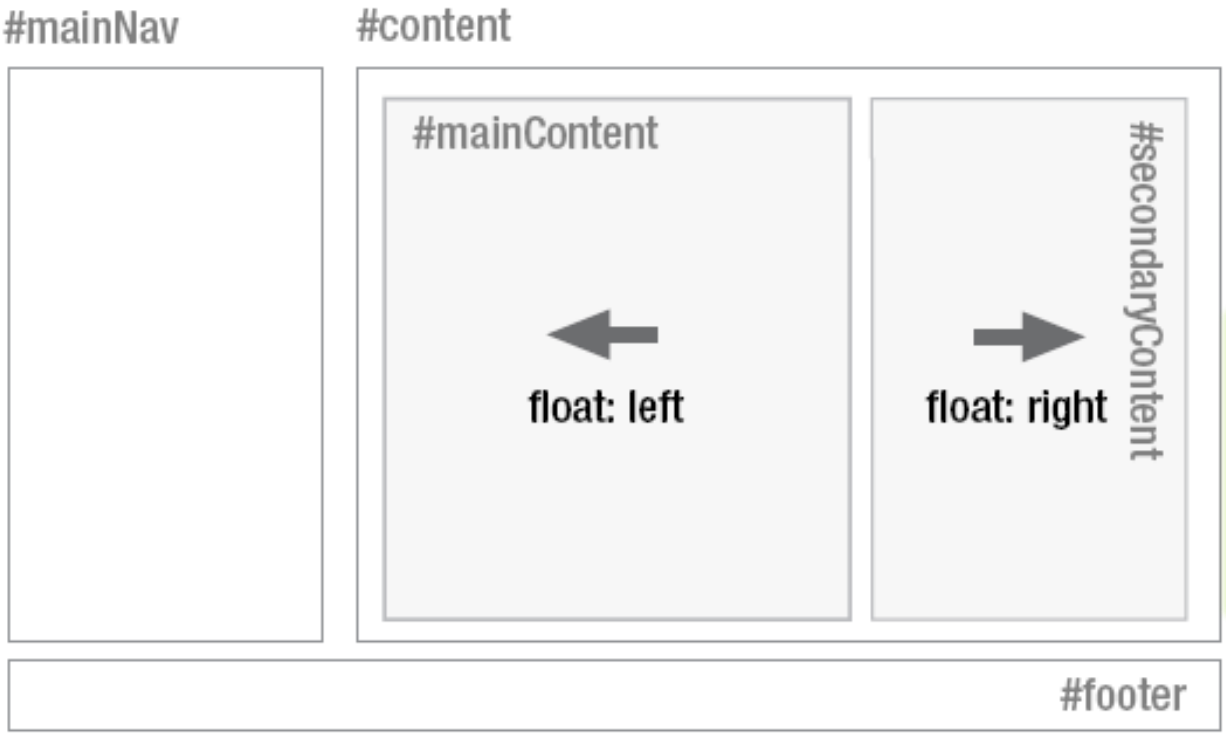
Three-column floated layout

- HTML framework
  - Two new divs inside the content div

```
<div id="content">
  <div id="mainContent">
     …
  </div>
  <div id="secondaryContent">
     …
  </div>
</div>
```

- Float the main content left and the secondary content right, inside the already floated content div
  - Divides the second content column in two, creating a three-column effect

Three-column floated layout

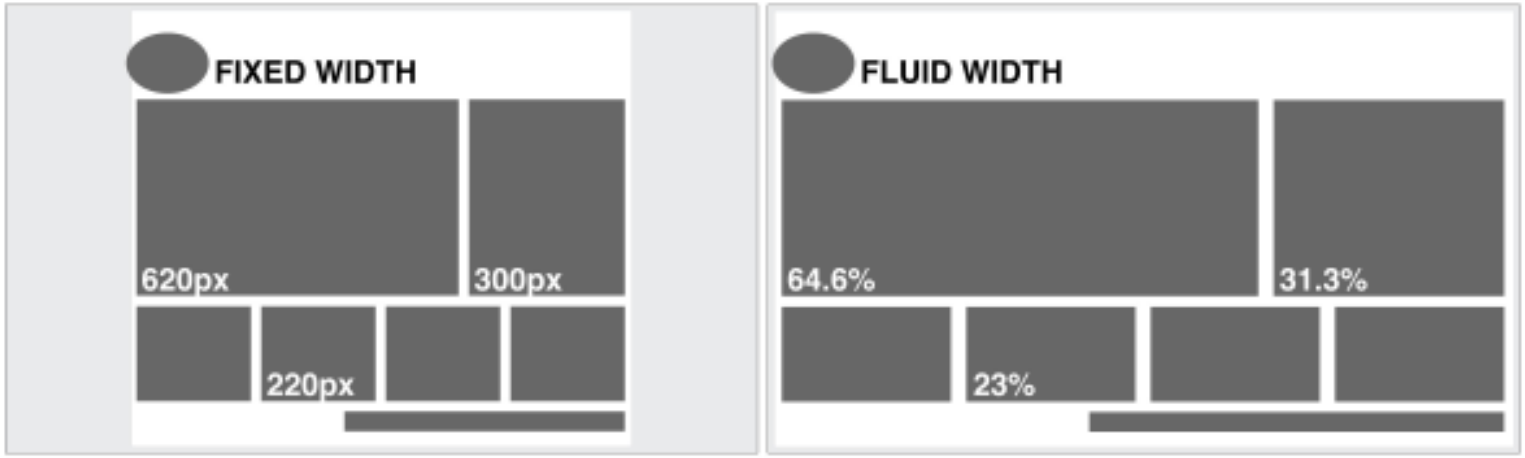



```
#mainContent {
width: 320px;
float: left; }
#secondaryContent {
width: 180px;
float: right; }
```

```
#secondaryContent h1, #secondaryContent h2,
     #secondaryContent p {
padding-left: 20px;
padding-right: 20px; }
```

Fixed-width, liquid, and elastic layout

- Different ways of defining column widths
- Different behavior: pros and cons



Fixed-width layout

- Column widths defined in pixels
- Very common: they give the developer more control over layout and positioning
- Downsides
  - Do not make good use of the available space: columns are always the same size no matter the window size
  - Usually work well with the browser default text size, but if you increase the text size a couple of steps, sidebars start running out of space and the line lengths get too short to comfortably read

Liquid layout

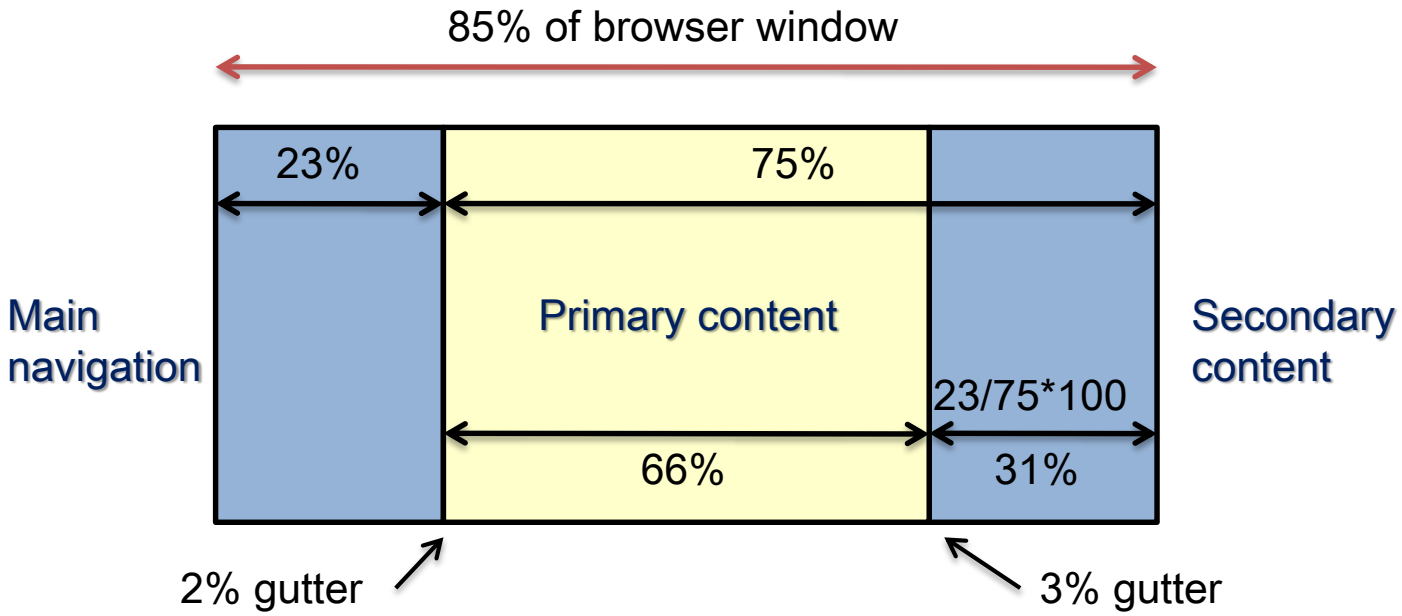
```
#wrapper {
width: 85%;
}
```

- Dimensions are set using percentages instead of pixels
  - Very efficient use of space
- If the design spans the entire width of the browser window, line lengths can become long and difficult to read
  - Make the wrapper span just a percentage, e.g. 85 percent
- Set the width of the navigation and content areas as a percentage of the wrapper width
  - 2-percent virtual gutter between the navigation and the wrapper to deal with any rounding errors and width irregularities that may occur

```
#wrapper {
width: 85%;
}
#mainNav {
width: 23%;
float: left;
}
#content {
width: 75%;
float: right;
}
```

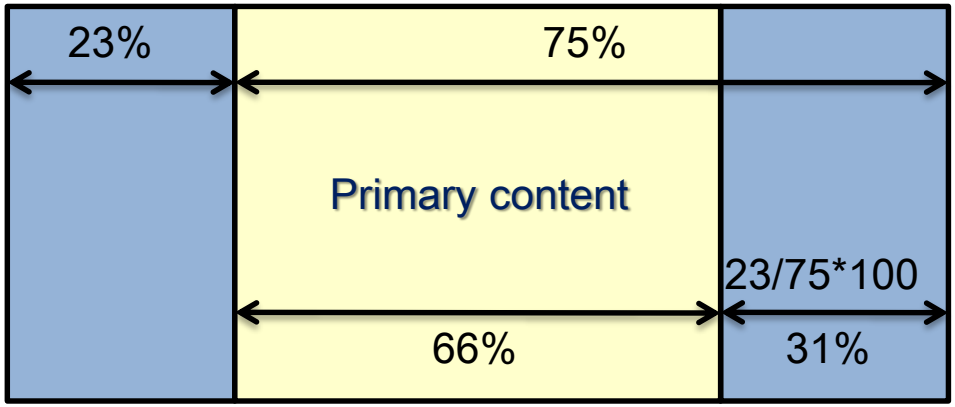
Liquid layout

- The widths of the content divs are based on the width of the content element and not the overall wrapper
  - Width of secondary content area = width of the main navigation area?



Liquid layout

- 3 columns liquid layout




```
#wrapper {
width: 85%;
}
#mainNav {
width: 23%;
float: left;
}
#content {
width: 75%;
float: right;
}
```

```
#mainContent {
width: 66%;
float: left;
}
#secondaryContent {
width: 31%;
float: right;
}
```

Elastic layout

- With liquid layouts
  - Line lengths can get uncomfortably long on large resolution monitors
  - Lines can become very short and fragmented in narrow windows or when the text size is increased a couple of steps
- In elastic layouts the width of elements is relative to the font size (ems) instead of the browser width
  - When the font size is increased the whole layout scales
- Allows to keep line lengths to a readable size
  - Particularly useful for people with reduced vision

Elastic layout

```
body {
font-size: 62.5%; }
```

- Trick to simplify design: set the base font size so that 1em roughly equals 10 pixels
  - The default font size on most browsers is 16 pixels
  - Ten pixels are 62.5 percent of 16 pixels
- Set the font size on the body to 62.5%
  - 1em now equals 10 pixels at the default font size
- Convert the fixed-width layout into an elastic layout by converting all the pixel widths to em widths

```
#wrapper {
width: 72em;
margin: 0 auto;
text-align: left; }
#mainNav {
width: 18em;
float: left; }
```

```
#content {
width: 52em;
float: right; }
#mainContent {
width: 32em;
float: left; }
#secondaryContent {
width: 18em;
float: right; }
```

Elastic-liquid hybrid

- Combines both elastic and liquid techniques
- Works by setting the widths in ems, then setting the maximum widths as percentages
- This layout will scale relative to the font size but will never get any larger than the width of the window

```
#wrapper {
width: 72em;
max-width: 100%;
margin: 0 auto;
text-align: left;
}
#mainNav {
width: 18em;
max-width: 23%;
float: left;
}
```

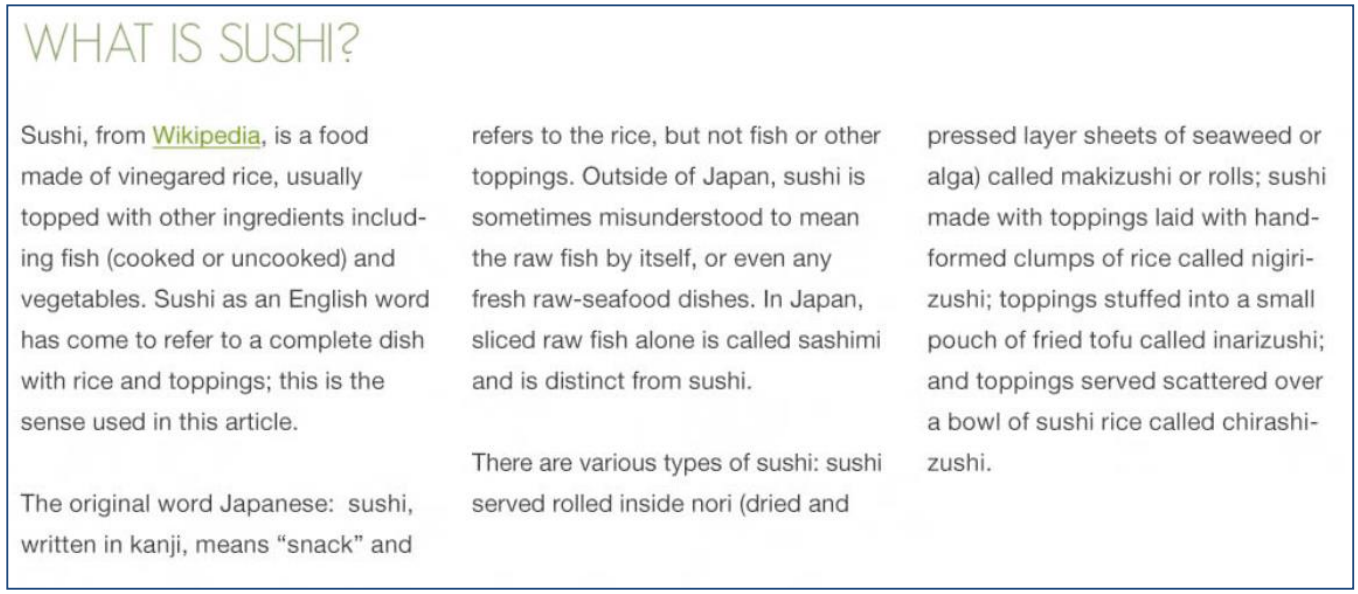
```
#content {
width: 52em;
max-width: 75%;
float: right;
}
#mainContent {
width: 32em;
max-width: 66%;
float: left;
}
#secondaryContent {
width: 18em;
max-width: 31%;
float: right;
}
```

Multi-column layout

- Novelty from CSS3
- Allows to get multi-column layouts without having to use multiple divs
- Different purpose, though!

```
.entry-content {
  column-count: 2;
  column-gap: 30px; }
```

```
.entry-content {
  column-width: 270px;
  column-gap: 30px; }
```



WHAT IS SUSHI?

Sushi, from Wikipedia, is a food made of vinegared rice, usually topped with other ingredients including fish (cooked or uncooked) and vegetables. Sushi as an English word has come to refer to a complete dish with rice and toppings; this is the sense used in this article.

The original word Japanese: sushi, written in kanji, means "snack" and refers to the rice, but not fish or other toppings. Outside of Japan, sushi is sometimes misunderstood to mean the raw fish by itself, or even any fresh raw-seafood dishes. In Japan, sliced raw fish alone is called sashimi and is distinct from sushi.

There are various types of sushi: sushi served rolled inside nori (dried and pressed layer sheets of seaweed or alga) called makizushi or rolls; sushi made with toppings laid with hand-formed clumps of rice called nigiri-zushi; toppings stuffed into a small pouch of fried tofu called inarizushi; and toppings served scattered over a bowl of sushi rice called chirashi-zushi.

Multi-column layout

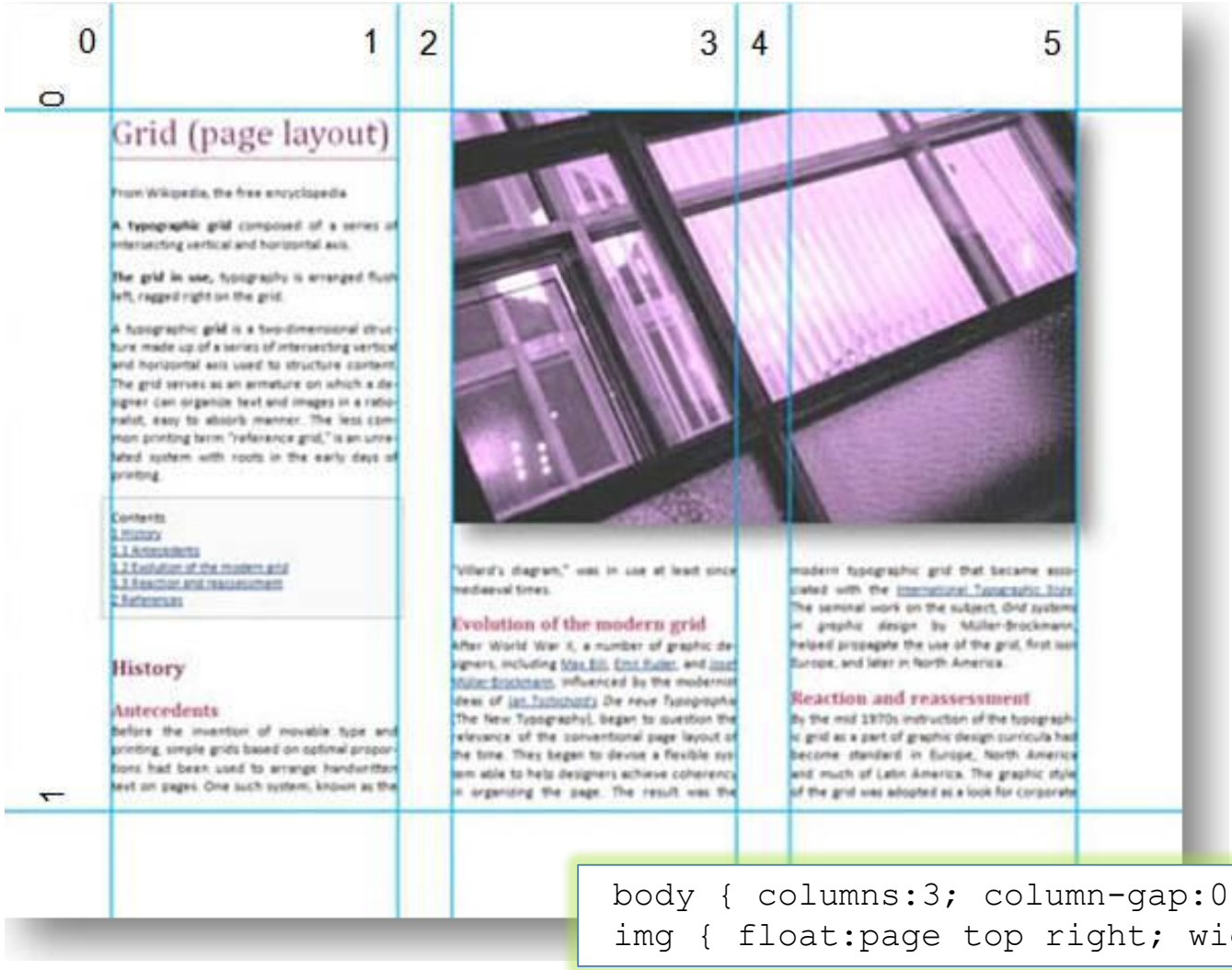
```
div
{ grid-columns:50% * * 200px; }
```

- Add one grid line in the middle of the div element, another one 200 pixels from the right, and another one in the middle of remaining space

```
div
{ grid-rows: 100px (30px 60px); }
```

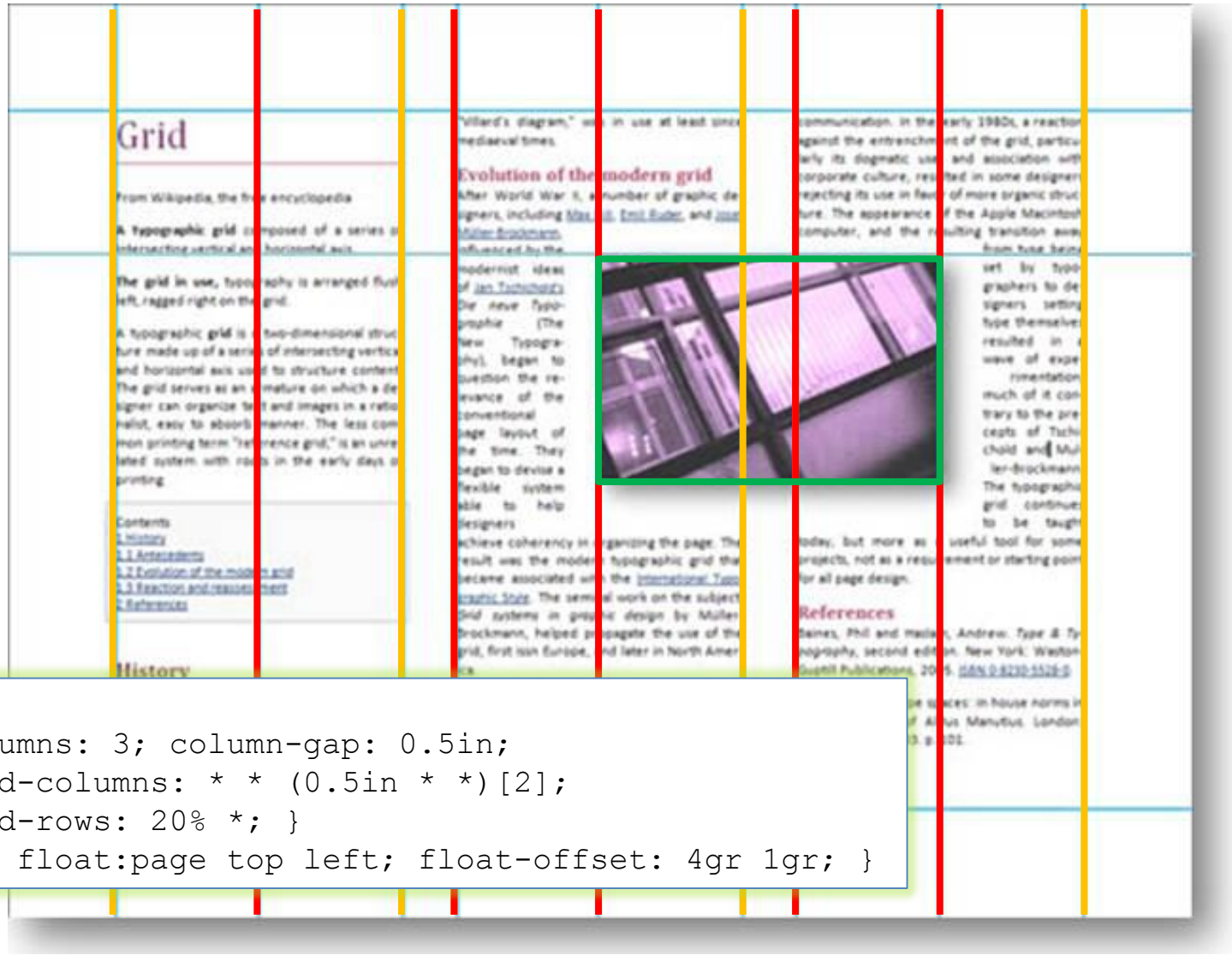
- Define a header row of 100 pixels, and add as many additional rows as necessary, alternating heights of 30 and 60 pixels

Example



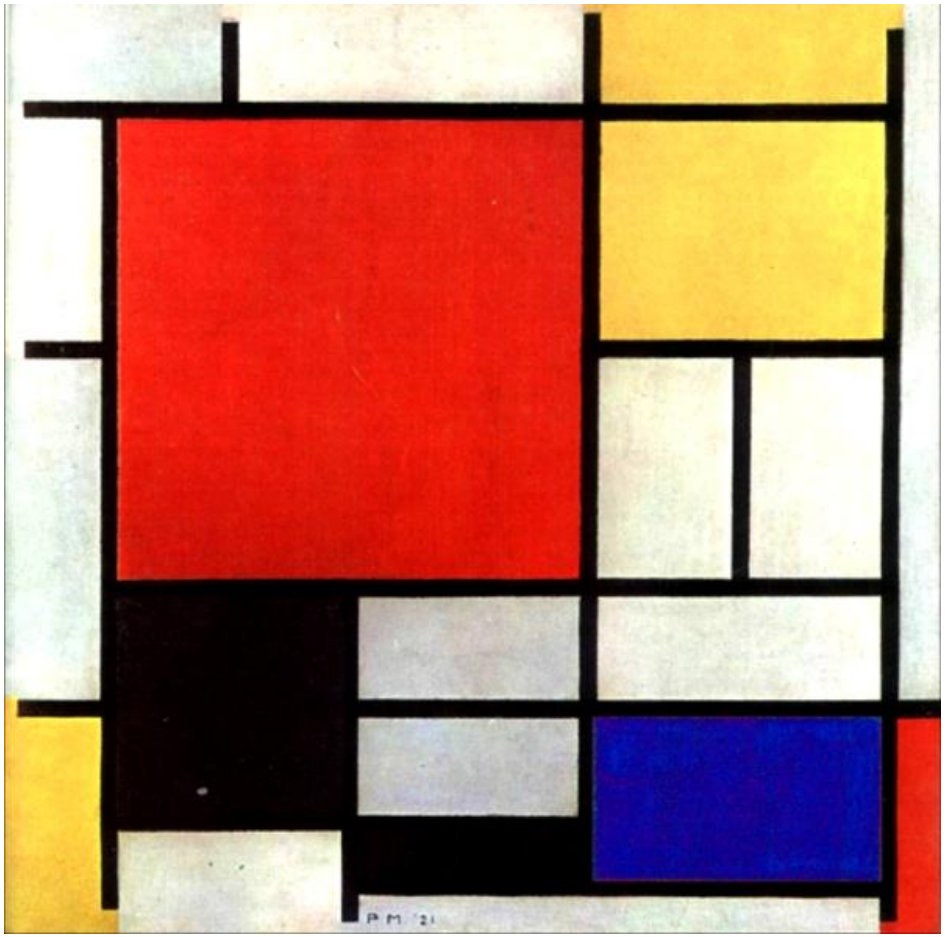

```
body { columns:3; column-gap:0.5in; }
img { float:page top right; width:3gr; }
```

Example




```
body
{ columns: 3; column-gap: 0.5in;
  grid-columns: * * (0.5in * *)[2];
  grid-rows: 20% *; }
img { float:page top left; float-offset: 4gr 1gr; }
```

Advanced layout: grid



Piet Mondrian, Composizione con grande piano rosso (1921)

Advanced layout: grid

Maki-zushi

The rice and seaweed rolls with fish and/or vegetables. There are also more specific terms for the rolls depending on the style.

Nigiri-zushi

The little fingers of rice topped with wasabi and a filet of raw or cooked fish or shellfish. Generally the most common form of sushi you will see.

Temaki-zushi

Also called a hand-roll. Cones of sushi rice, fish and vegetables wrapped in seaweed. It is very similar to maki.

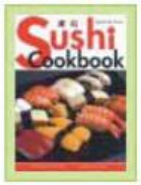
WHAT IS SUSHI?

Beginning as a method of preserving fish centuries ago, sushi has evolved into an artful, unique dining experience. In its earliest form, dried fish was placed between two pieces of vinegared rice as a way of making it last. The nori (seaweed) was added later as a way to keep one's fingers from getting sticky.

Technically, the word `sushi' refers to the rice, but colloquially, the term is used to describe a finger-size piece of raw fish or shellfish on a bed of vinegared rice or simply the consumption or raw fish in the Japanese style (while sushi is not solely a Japanese invention, these days, the Japanese style is considered the de facto serving standard).

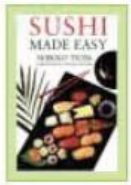
Sashimi

Sashimi is raw fish served sliced, but as-is. That means no rice bed or roll, but it is often served alongside daikon and/or shiso. This is my favorite style as you really get the flavor of the fish..



QUICK & EASY SUSHI COOKBOOK
This book has great pictures, however it is not as complete as Sushi Made Easy.

SUSHI FOR PARTIES: MAKI-ZUSHI AND NIGIRI-ZUSHI
This book also has great pictures, with advanced maki (cut roll) making techniques.



SUSHI MADE EASY
A very decent all-around book for the money.

Advanced layout: grid

Maki-zushi

a

The rice and seaweed rolls with fish and/or vegetables. There are also more specific terms for the rolls depending on the style.

a

Nigiri-zushi

c

The little fingers of rice topped with wasabi and a filet of raw or cooked fish or shellfish. Generally the most common form of sushi you will see.

Temaki-zushi

d

Also called a hand-roll. Cones of sushi rice, fish and vegetables wrapped in seaweed. It is very similar to maki.

WHAT IS SUSHI?

e

Beginning as a method of preserving fish centuries ago, sushi has evolved into an artful, unique dining experience. In its earliest form, dried fish was placed between two pieces of vinegared rice as a way of making it last. The nori (seaweed) was added later as a way to keep one's fingers from getting sticky.

e

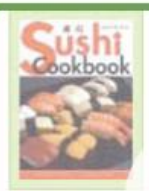
Technically, the word 'sushi' refers to the rice, but colloquially, the term is used to describe a finger-size piece of raw fish or shellfish on a bed of vinegared rice or simply the consumption of raw fish in the Japanese style (while sushi is not solely a Japanese invention, these days, the Japanese style is considered the de facto serving standard).

g

Sashimi

i

Sashimi is raw fish served sliced, but as-is. That means no rice bed or roll, but it is often served alongside daikon and/or shiso. This is my favorite style as you really get the flavor of the fish..



j

QUICK & EASY SUSHI COOKBOOK

This book has great pictures, however it is not as complete as Sushi Made Easy.

k

SUSHI FOR PARTIES, MAKI-ZUSHI AND NIGIRI-ZUSHI

This book also has great pictures, with advanced maki (cut roll) making techniques.

l

SUSHI MADE EASY

A very decent all-around book for the money.

Advanced layout: grid

- It is possible to define a grid in which content can flow or be placed, or that remain empty
- There are three ways to define a grid
  - Explicit grid: defined with ‘grid-columns’ and ‘grid-rows’ properties
  - Natural grid: automatically created by elements with a natural grid structure (multi-column elements and tables)
  - Default grid: all other block elements define a single-cell grid

Example

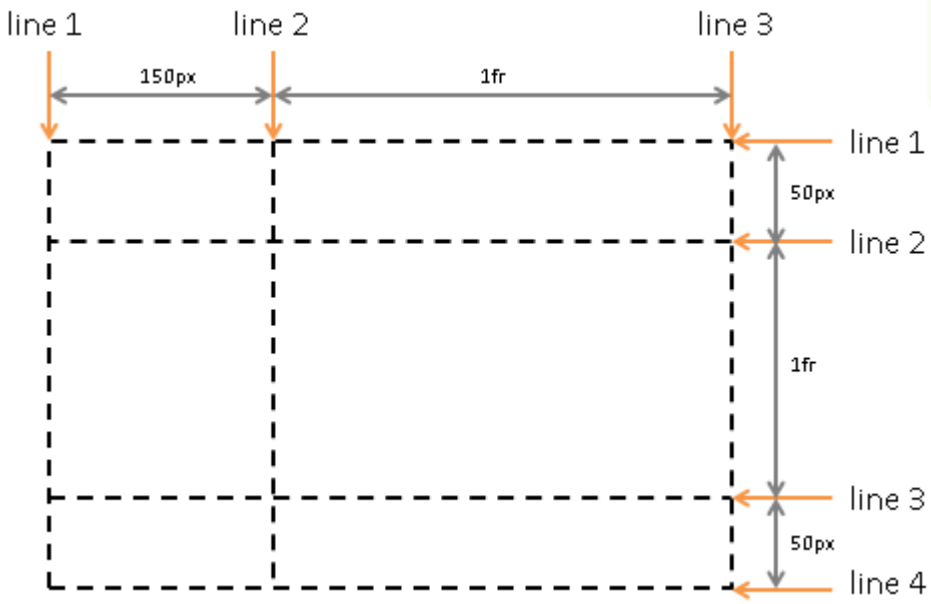
- Classic three-column layout

```
<section>
  <header>Title</header>
  <nav>Menu</nav>
  <article>Content</article>
  <aside>Notes</aside>
  <footer>Footer</footer>
</section>
```

Virtual grid

- First step: describe the grid tracks – rows and columns inside the grid

```
#grid {
  grid-columns: 150px 1fr;
  /* two columns */
  grid-rows: 50px 1fr 50px
  /* three rows */ }
```



fr = fraction values

- new unit applicable to grid-rows and grid-columns properties

Virtual grid

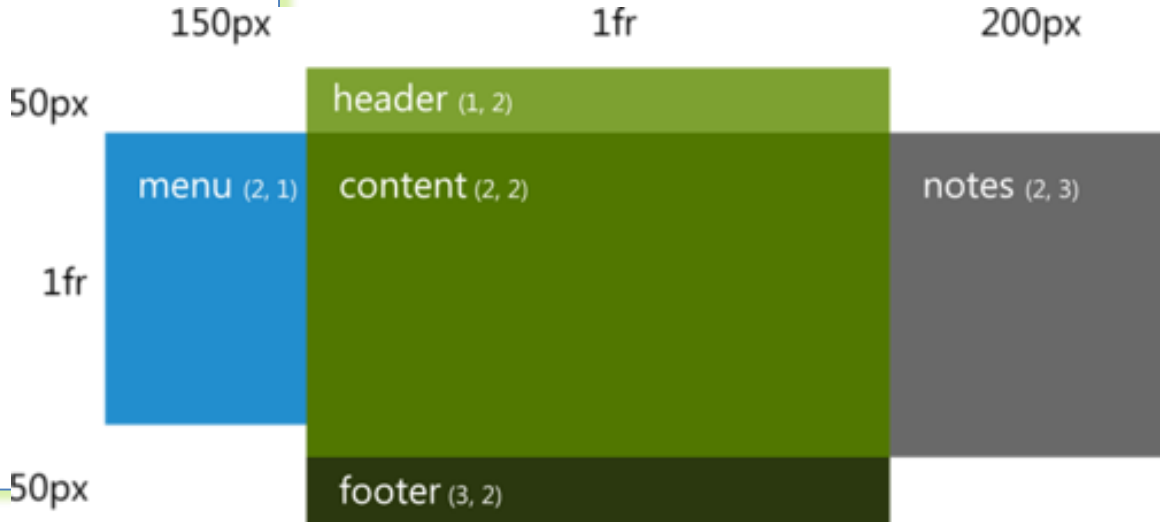
- Second step: to set how exactly the element will be placed on a grid it is necessary to specify to which line (both vertical and horizontal) it will be attached


```
#item {
  grid-column: 2;
  grid-row: 2; }
```

Example

```
section {
  display: grid;
  grid-columns: 150px 1fr 200px;
  grid-rows: 50px 1fr 50px; }
section header {
  grid-column: 2;
  grid-row: 1; }
section nav {
  grid-column: 1;
  grid-row: 2; }
section article {
  grid-column: 2;
  grid-row: 2; }
section aside {
  grid-column: 3;
  grid-row: 2; }
section footer {
  grid-column: 2;
  grid-row: 3; }
```

- fr = fraction values
  - new unit applicable to grid-rows and grid-columns properties

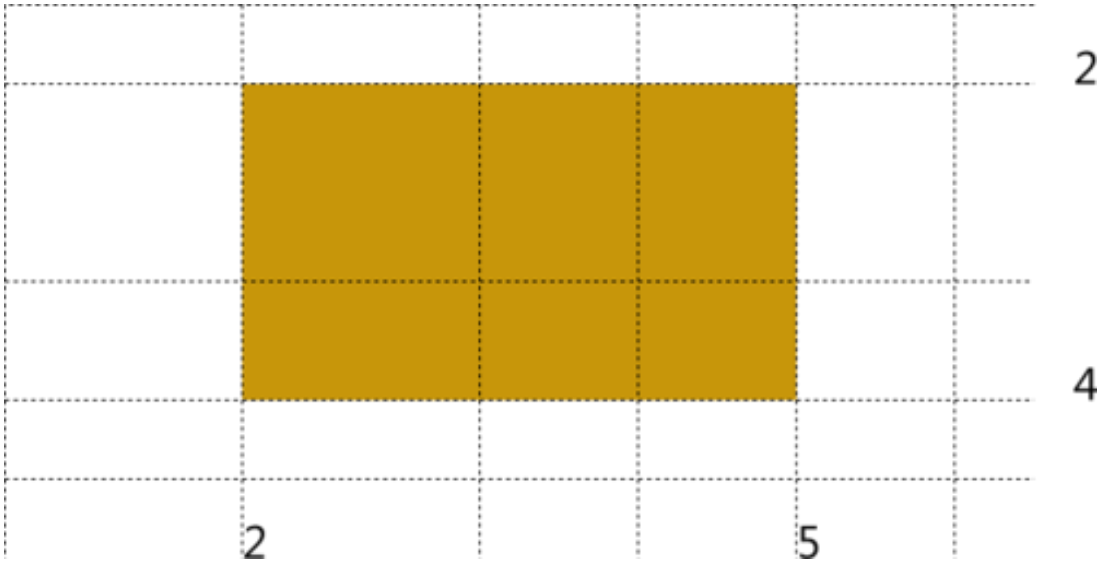


Fraction values

- The distribution of fractional space occurs after all or content-based row and column sizes have reached their maximum
- The total size of the rows or columns is then subtracted from the available space and the remainder is divided proportionately among the fractional rows and columns

- Auto: height or width depends on content

Expansion on several cells




```
#item {
  grid-column: 2;
  grid-column-span: 3;
  grid-row: 2;
  grid-row-span: 2; }
```

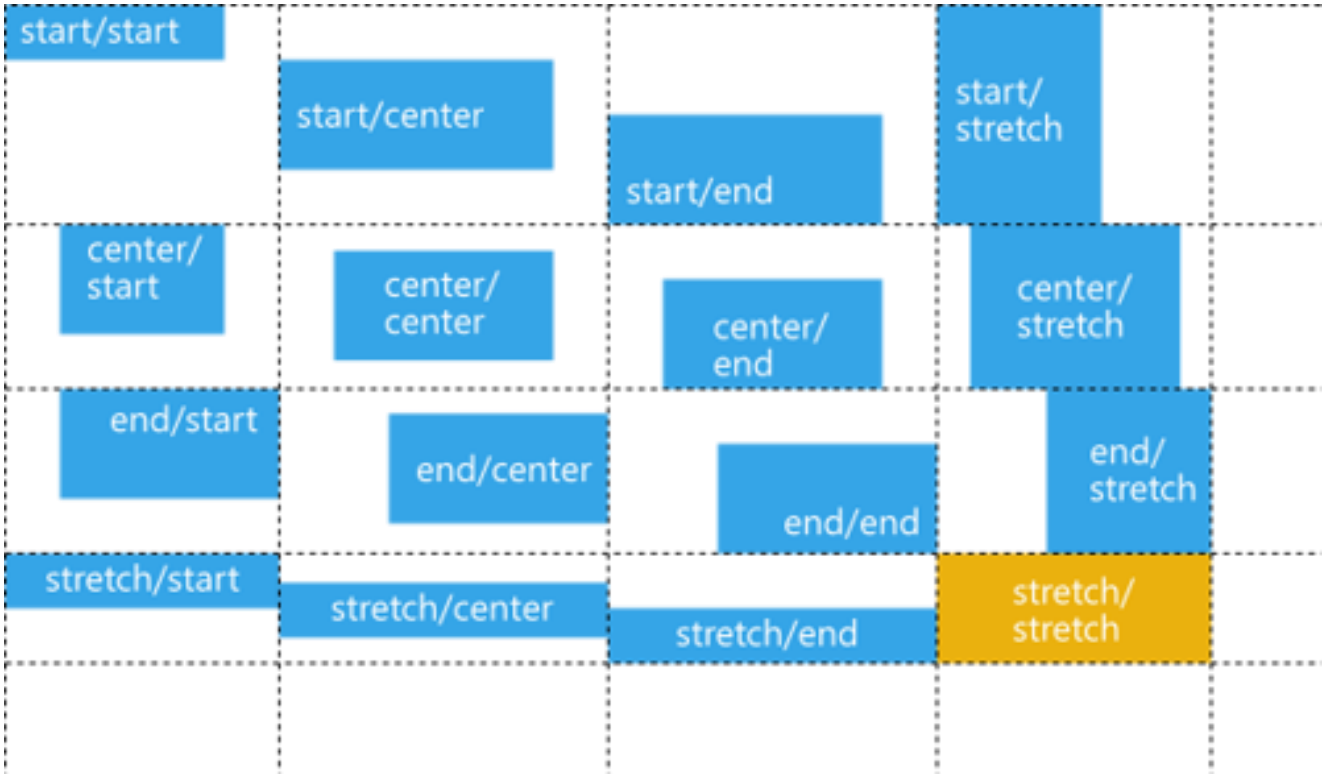
Repeated tracks

- Compact way to describe a repetitive grid layout

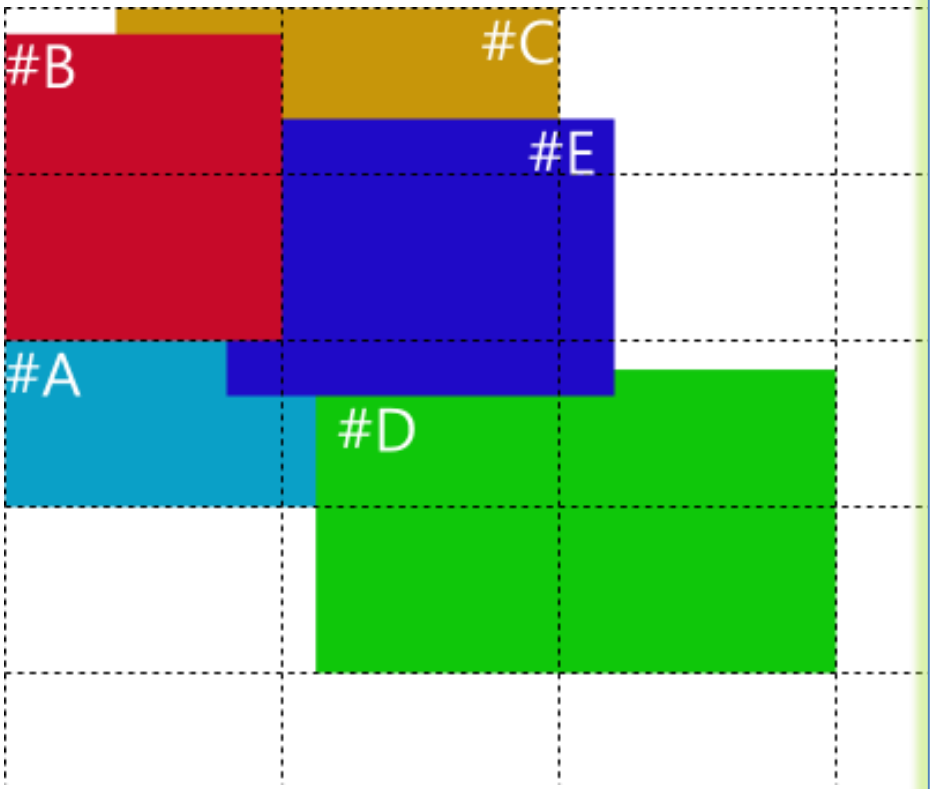
```
#grid {
  display: grid;
  grid-columns: 24px (120px 24px)[4];
  grid-rows: (1fr 24px)[3]; }
```

Binding of the elements

- Controlled by the properties grid-column-align and grid-row-align
  - Values: start, end, center, stretch



Layers control




```
#grid {
  display: grid;
  grid-columns: (1fr)[3];
  grid-rows: (1fr)[4]; }
#A {
  grid-column:1; grid-row:3;
  grid-column-span:2; }
#B {
  grid-column:1; grid-row:1;
  grid-row-span:2;
  z-index:10;
  margin-top:10px; }
#C {
  grid-column:1; grid-row:1;
  grid-column-span:2;
  margin-left:50px; }
#D {
  grid-column:2; grid-row:3;
  grid-row-span:2;
  grid-column-span:2;
  margin:10px 0 0 10px; }
#E {
  grid-column:2; grid-row:2;
  z-index:5;
  margin: -20px; }
```

CSS MEDIA QUERIES AND RESPONSIVE DESIGN

Media queries

- A media query consists of a media type and can contain one or more expressions, which resolve to either true or false
- The result of the query is true if the specified media type matches the device the document is being displayed on and if all expressions in the media query are true
- When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules

```
@media not|only mediatype and (expressions)
{
    CSS-Code;
}
```

- Unless you use the not or only operators, the media type is optional and the “all” media type is implied

Media queries

- It is possible to have different stylesheets for different media

```
<link rel="stylesheet"
      media="mediatype and|not|only (expressions)"
      href="print.css">
```

- Examples

```
<link rel="stylesheet" type="text/css"
href="a.css" media="screen and (color)">
```

Media type

Keyword

Expression

Media feature

```
media="(color)"
```

Media types

- Can be used to specify how a document is presented on different media

CSS3 Media Types

Value	Description
all	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

- CSS 2.1 defined ten media types (all, aural, braille, embossed, handheld, print, projection, screen, tty, tv), but they never got a real support by devices, other than the print media type, and they are now deprecated

Media queries

- Media queries look at the capability of the device, and can be used to check many things, such as:
  - Width and height of the viewport
  - Width and height of the device
  - Orientation (is the tablet/phone in landscape or portrait mode?)
  - Resolution
  - … and much more
- List of supported media features
  - http://www.w3schools.com/cssref/css3_pr_mediaquery.asp

Media features (some)

Value	Description
aspect-ratio	The ratio between the width and the height of the viewport
color	The number of bits per color component for the output device
color-index	The number of colors the device can display
device-aspect-ratio	The ratio between the width and the height of the device
device-height	The height of the device, such as a computer screen
device-width	The width of the device, such as a computer screen
grid	Whether the device is a grid or bitmap
height	The viewport height
max-aspect-ratio	The maximum ratio between the width and the height of the display area
max-color	The maximum number of bits per color component for the output device
max-color-index	The maximum number of colors the device can display
max-device-aspect-ratio	The maximum ratio between the width and the height of the device
max-device-height	The maximum height of the device, such as a computer screen
max-device-width	The maximum width of the device, such as a computer screen

Media queries

- Most media features accept “min-” or “max-” prefixes

```
media="screen and (min-height: 20em)"
```

- Media features can often be used without a value

```
media="screen and (color)"
```

- Media features only accept single values: one keyword, one number, or a number with a unit identifier

```
(orientation: portrait)
(min-width: 20em)
(min-color: 2)
(device-aspect-ratio: 16/9)
```

Media queries

- Examples

```
media="not screen and (color)">
```

applied to all devices except those with color screens

```
media="only screen and (color)">
```

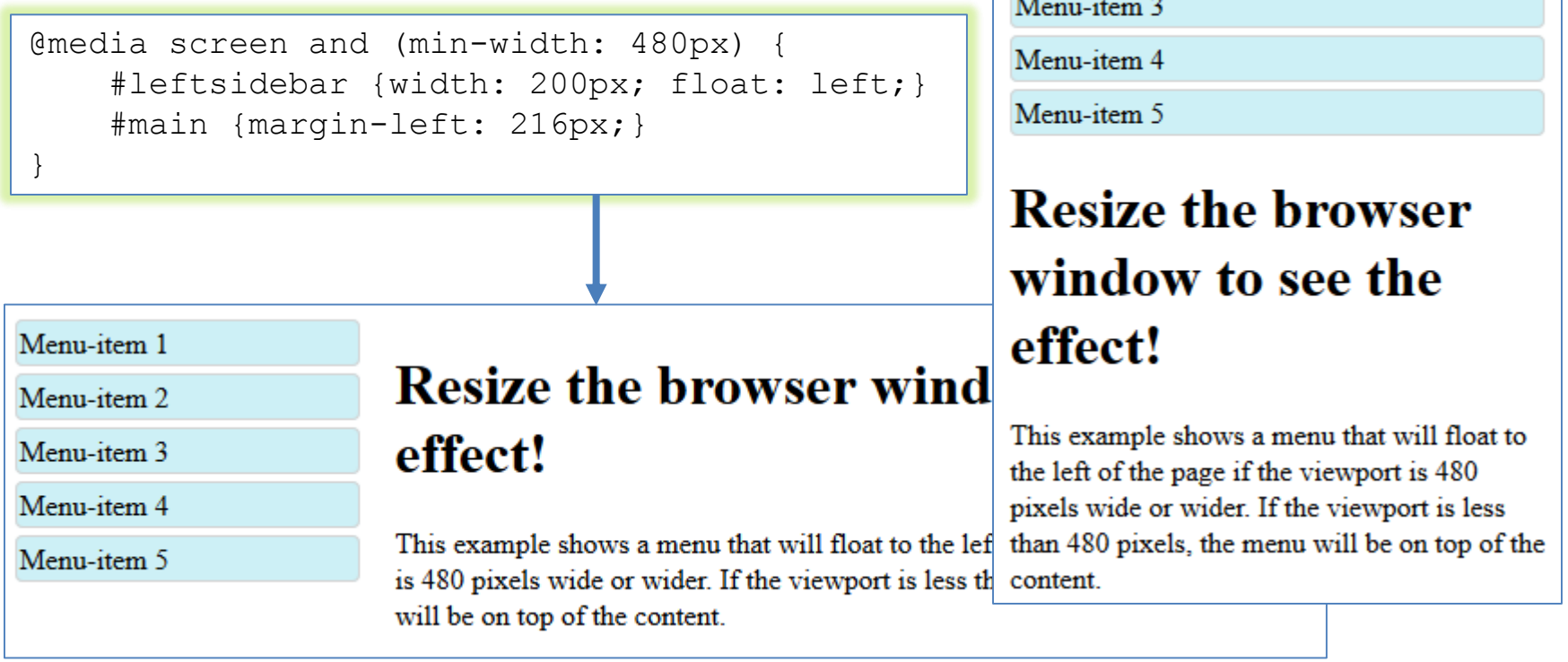
applied only to all devices with color screens

MediaQueryDemo.htm

https://www.w3schools.com/css/tryit.asp?filename=trycss3_media_queries2

Example

```
@media screen and (min-width: 480px) {
    #leftsidebar {width: 200px; float: left;}
    #main {margin-left: 216px;}
}
```

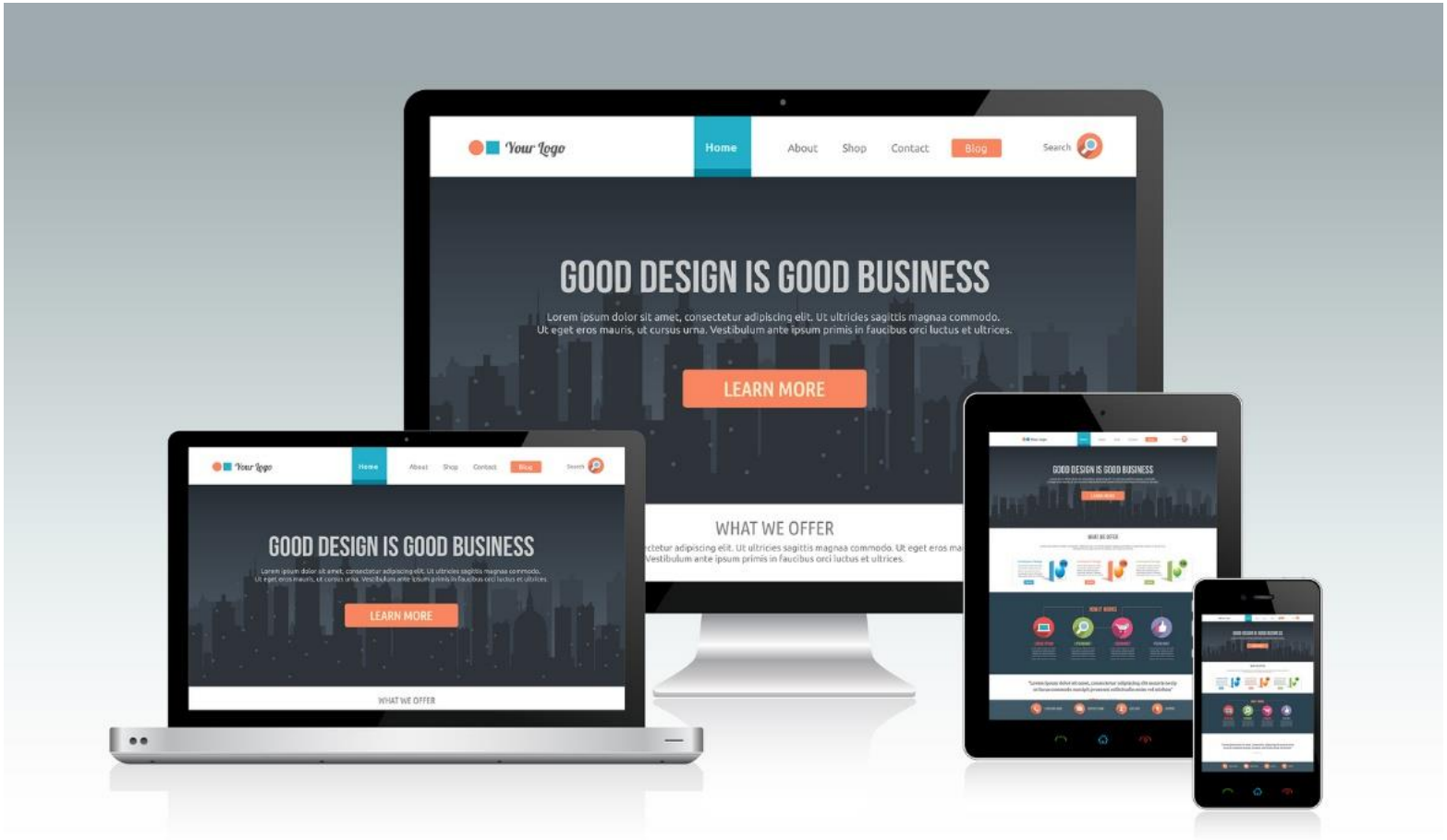


https://www.w3schools.com/css/tryit.asp?filename=trycss3_media_queries2

Responsive web design

- Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation
- Responsive web design makes your web page look good on all devices (desktops, tablets, and phones)
  - Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device
- Responsive web design uses only HTML and CSS
  - to resize, hide, shrink, enlarge, or move the content to make it look good on any screen
- The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries

Responsive web design



The viewport

- The viewport is the user's visible area of a web page
- The viewport varies with the device
  - smaller on a mobile phone than on a computer screen
- Before tablets and mobile phones, web pages were designed only for computer screens, and web pages had a static design and a fixed size
- When users started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport
  - To fix this, browsers on those devices scaled down the entire web page to fit the screen.
  - This was not perfect…

The viewport

- The HTML5 <meta> tag allows to take control over the viewport
- A <meta> viewport element gives the browser instructions on how to control the page dimensions and scaling
- All web pages should include

```
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

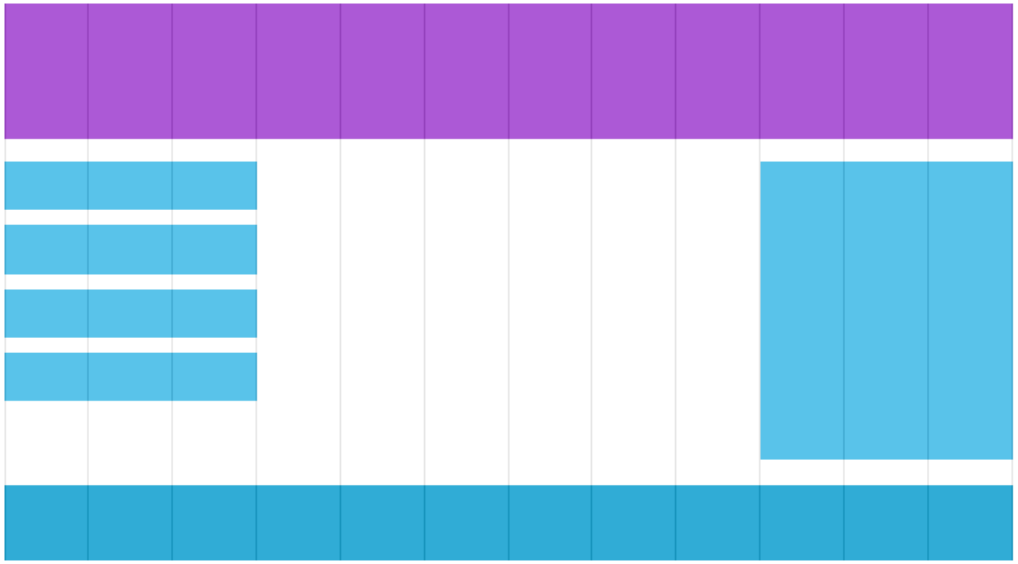
- The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device)
- The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser

Size content to the viewport

- Users are used to scroll websites vertically on both desktop and mobile device
- If the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience
- Some rules
  - Do NOT use large fixed width elements (e.g. images): remember to adjust this content to fit within the width of the viewport
  - Do NOT let the content rely on a particular viewport width to render well: screen dimensions and width in CSS pixels vary widely between devices
  - Use CSS media queries to apply different styling for small and large screens: consider using relative width values(percentages)

Grid-view

- Many web pages are based on a grid-view, i.e. the page is divided into columns
  - Very helpful when designing web pages: easier to place elements on the page
- A responsive grid-view often has 12 columns, a total width of 100%, and will shrink and expand as you resize the browser window



Building a responsive grid-view

- All HTML elements should have the box-sizing property set to border-box
  - This makes sure that the padding and border are included in the total width and height of the elements
- Responsive grid-view with 12 columns
  - Percentage for one column: 100% / 12 columns = 8.33%
- Then we make one class for each of the 12 columns
  - class = “col-n”, where n defines how many columns the section should span

```
* {
      box-sizing: border-box;
}
```

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

Building a responsive grid-view

- All these columns should float to the left, and have a padding
- Each row should be wrapped in a
  - The number of columns inside a row should always add up to 12

```
[class*="col-"] {
    float: left;
    padding: 15px;
}
```

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
</div>
```

- The columns inside a row are all floating to the left
  - The other elements will be placed as if the columns do not exist
  - Add a style that clears the flow

```
.row::after {
    content: "";
    clear: both;
    display: table;
}
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_styles

Responsive design with media queries

- Breakpoint at 768px
- When the screen (browser window) gets smaller than 768px, each column should have a width of 100%

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_breakpoints

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
    /* For mobile phones: */
    [class*="col-"] {
        width: 100%;
    }
}
```

Design for mobile first

- To make the page display faster on smaller devices

```
/* For mobile phones: */
[class*="col-"] {
    width: 100%;
}
@media only screen and (min-width: 768px) {
    /* For desktop: */
    .col-1 {width: 8.33%;}
    .col-2 {width: 16.66%;}
    .col-3 {width: 25%;}
    .col-4 {width: 33.33%;}
    .col-5 {width: 41.66%;}
    .col-6 {width: 50%;}
    .col-7 {width: 58.33%;}
    .col-8 {width: 66.66%;}
    .col-9 {width: 75%;}
    .col-10 {width: 83.33%;}
    .col-11 {width: 91.66%;}
    .col-12 {width: 100%;}
}
```

More breakpoints

- For tablets: one more media query (at 600px), and a set of new classes for devices larger than 600px (but smaller than 768px)

```
@media only screen and (min-width: 600px)
{
    /* For tablets: */
    .col-m-1 {width: 8.33%;}
    .col-m-2 {width: 16.66%;}
    .col-m-3 {width: 25%;}
    .col-m-4 {width: 33.33%;}
    .col-m-5 {width: 41.66%;}
    .col-m-6 {width: 50%;}
    .col-m-7 {width: 58.33%;}
    .col-m-8 {width: 66.66%;}
    .col-m-9 {width: 75%;}
    .col-m-10 {width: 83.33%;}
    .col-m-11 {width: 91.66%;}
    .col-m-12 {width: 100%;}
}
```

```
<div class="row">
<div class="col-3 col-m-3">...</div>
<div class="col-6 col-m-9">...</div>
<div class="col-3 col-m-12">...</div>
</div>
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_col-s

Orientation: portrait / landscape

- Media queries can also be used to change layout of a page depending on the orientation of the browser
- Set of CSS properties that will only apply when the browser window is wider than its height
  - "Landscape" orientation

```
body {
    background-color: lightgreen;
}

@media only screen and (orientation: landscape) {
    body {
        background-color: lightblue;
    }
}
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_mediaquery_orientation

Images

- If the width property is set to 100%, the image will be responsive and scale up and down

```
img {
    width: 100%;
    height: auto;
}
```

- If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size

```
img {
    max-width: 100%;
    height: auto;
}
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_image

Different images for different devices

- A large image can be perfect on a big computer screen, but useless on a small device
  - Why load a large image when you have to scale it down anyway?
- To reduce the load, or for any other reasons, you can use media queries to display different images on different devices

```
/* For devices smaller than 400px: */
body {
    background-image: url('img_smallflower.jpg'); }

/* For devices 400px and larger: */
@media only screen and (min-device-width: 400px) {
    body {
        background-image: url('img_flowers.jpg'); }
}
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_image_mediaq

The <picture> element

- HTML5 introduced the <picture> element
  - Similar to the <audio> and <video> elements: allows to define different sources, and the first source that fits the preferences is the one being used
- The media attribute is optional, and accepts the media queries

```
<picture>
  <source srcset="img_smallflower.jpg" media="(max-width: 400px)">
  <source srcset="img_flowers.jpg">
  <img src="img_flowers.jpg" alt="Flowers">
</picture>
```

Frameworks for responsive web design

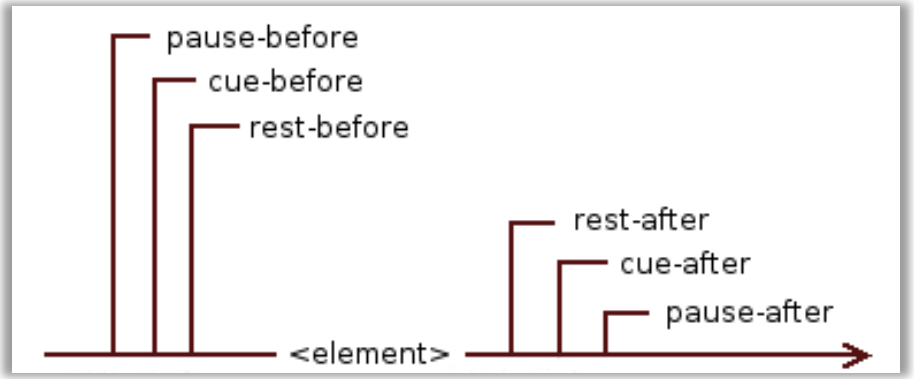
- W3.CSS
  - https://www.w3schools.com/css/css_rwd_frameworks.asp
  - Many templates available
- Bootstrap
  - uses HTML, CSS and jQuery
  - http://getbootstrap.com/
  - https://www.w3schools.com/bootstrap/default.asp

Aural Style Sheets

http://www.w3schools.com/cssref/css_ref_aural.asp

- Allows to specify the speech style of screen readers by controlling various aspects of the speech, such as:
  - Voice-volume
    Set a volume using a number from 0 to 100 (0 being silence), percentages or a keyword (silent, x-soft, soft, medium, loud and x-loud)
  - Voice-family
    Set specific types of voices and voice combinations (as you do with fonts)
  - Voice-balance
    Control which channel sound comes from (if the user's sound system supports stereo)
  - Speak
    Instruct the screen reader to spell out particular words, digits or punctuation. Available keywords are none, normal, spell-out, digits, literal-punctuation, no-punctuation and inherit

Aural Style Sheets



- Pauses and rests
  Set a pause or rest before or after an element's content is spoken. You can use either time units (for example, "2s" for 2 seconds) or keywords (none, x-weak, weak, medium, strong and x-strong)
- Cues
  Use sounds to delimit particular elements and control their volume
- Voice-rate
  Control the speed at which elements are spoken. This can be defined as a percentage or keyword: x-slow, slow, medium, fast and x-fast
- Voice-stress
  Indicate any emphasis that should be applied, using different keywords: none, moderate, strong and reduced

Media type speech

- Examples

```
@media speech
{ body { voice-family: Paul } }
```

```
h2
{ voice-family: female;
  voice-balance: left;
  voice-volume: soft;
  cue-after: url(sound.au);
}
```

Tells a screen reader to read all h2 tags in a female voice, from the left speaker, in a soft tone and followed by a particular sound

```
h1 { voice-family: announcer, old male }
p.part.romeo { voice-family: romeo, young male }
p.part.juliet { voice-family: juliet, female }
p.part.mercutio { voice-family: male 2 }
p.part.tybalt { voice-family: male 3 }
p.part.nurse { voice-family: child female }
```

License

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 3,0)” License.
- You are free:
  - to Share - to copy, distribute and transmit the work
  - to Remix - to adapt the work
- Under the following conditions:
  -  Attribution - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  -  Noncommercial - You may not use this work for commercial purposes.
  -  Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit http://creativecommons.org/license/by-nc-sa/3.0/