CSS: Cascading Style Sheets

BASICS, SELECTORS, BOX MODEL, PAGE LAYOUT

POLITECNICO DI TORINO

Laura Farinetti - DAUIN

e-Lite

Summary

- Introduction
- CSS syntax
- CSS selectors
- CSS cascading
- CSS box model
- CSS positioning schemes
- Page layout with CSS float
- CSS Flexbox

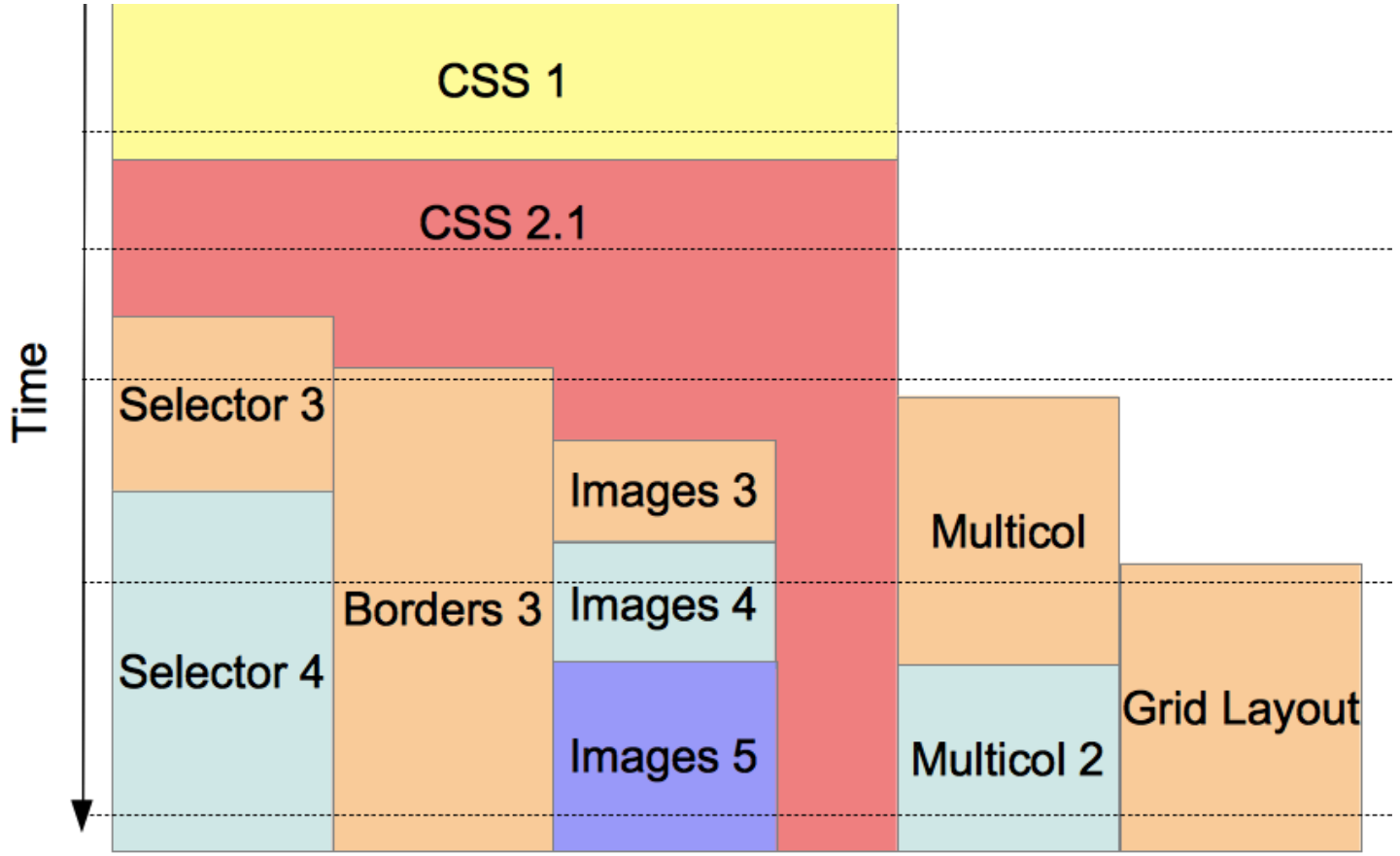
Cascading Style Sheets

- 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

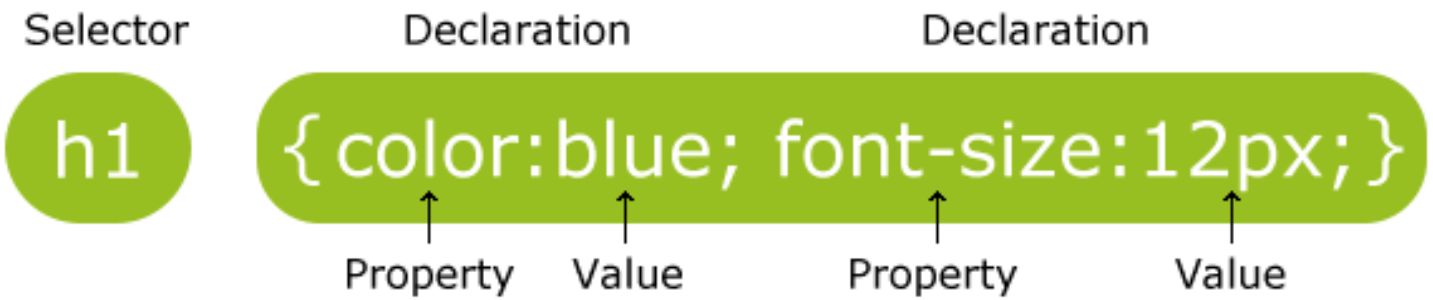


https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

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

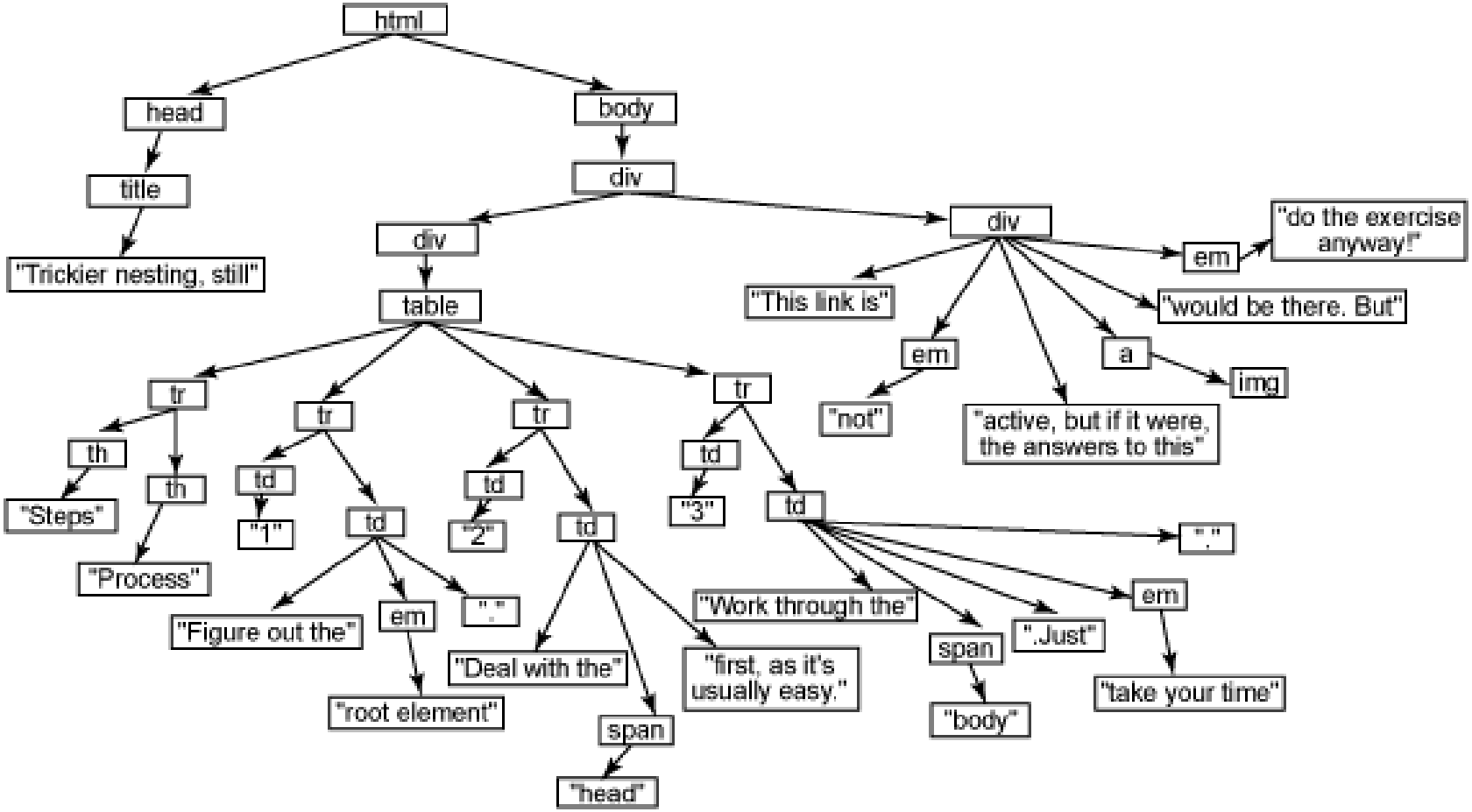
```
<html lang="en">
<head>
    <title>Trickier nesting, still</title>
</head>
<body>
    <div>
        <div>
            <tabl
                <
                <
                <
                <
            </tab
        </div>
        <div>
            This link is <em>not</em> active, but it it 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!*

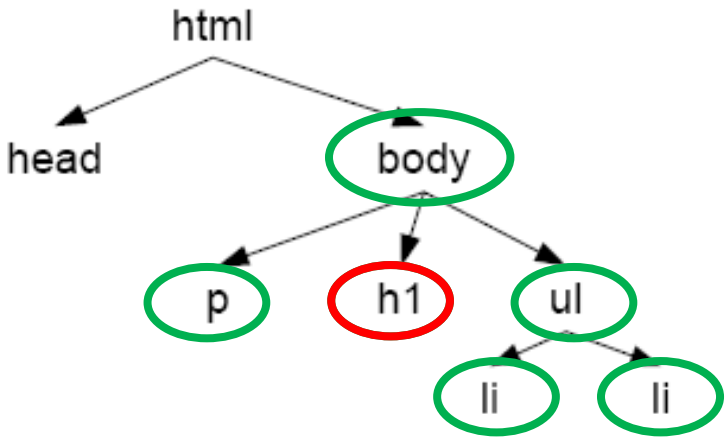
HTML structure

- 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*.

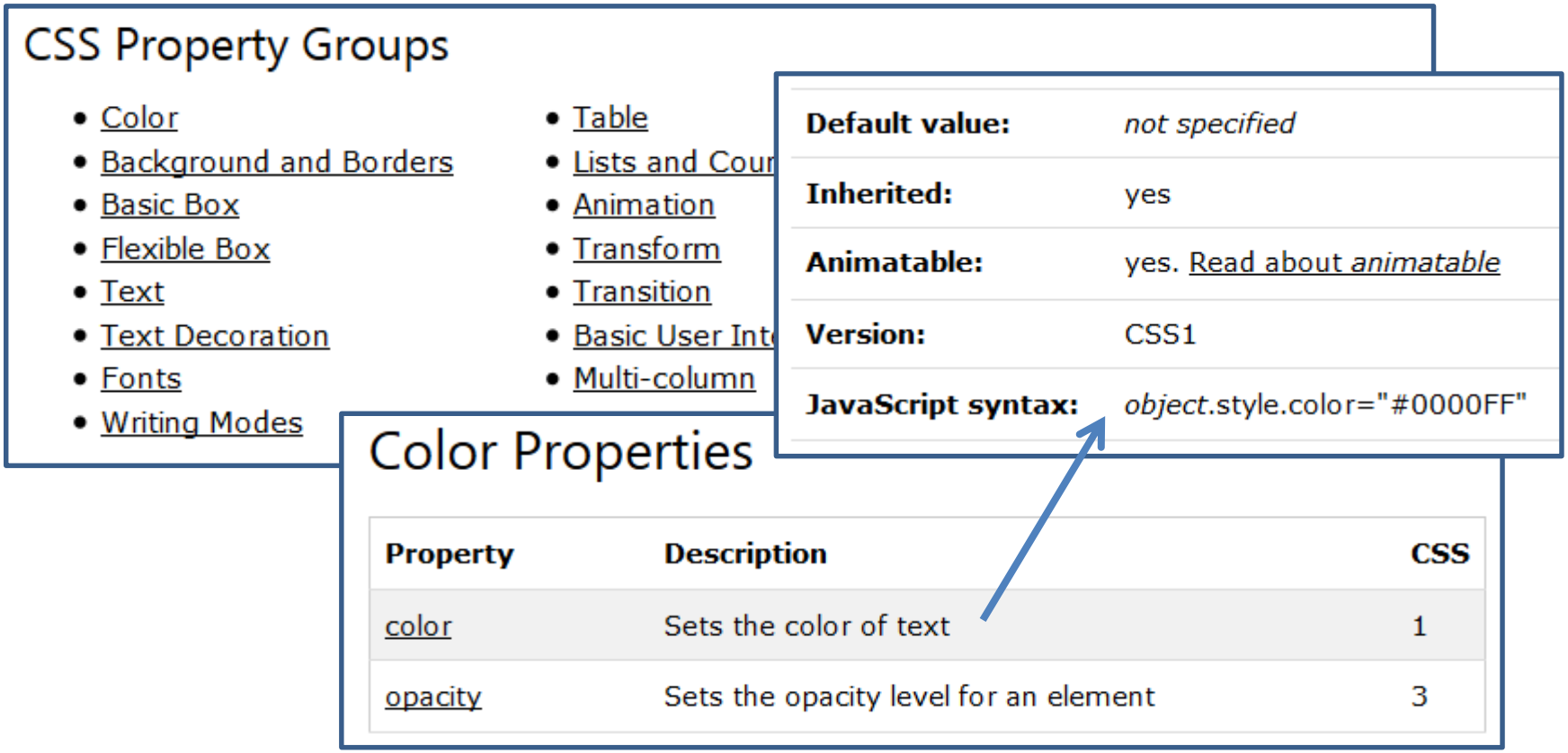
This link is *not* active, but it it were, the answer to this  would be there. But *do the exercise anyway!*

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

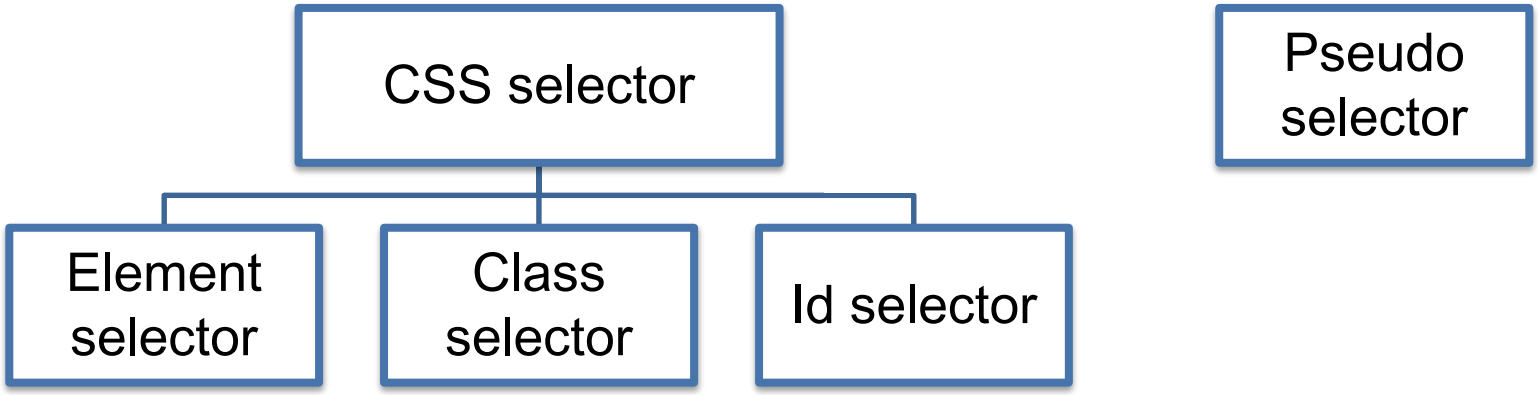
- http://www.w3schools.com/cssref/



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 instances of a specific element in a document
- 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

"BLOCK BOX"

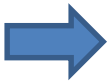
"INLINE BOXES"

Div element

"BLOCK BOX"

- 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

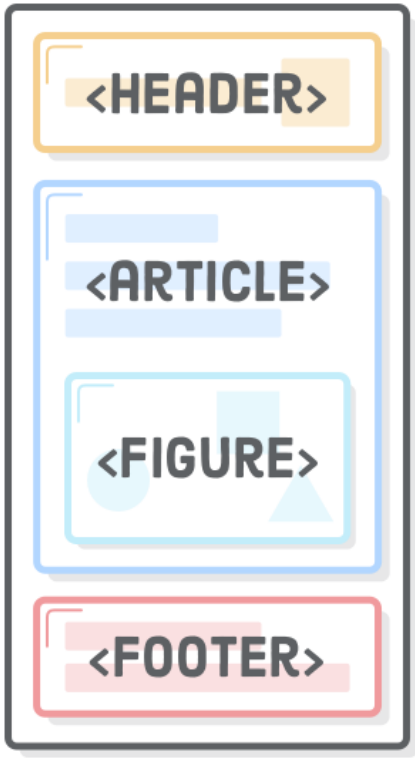
- Used to group or identify inline elements

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

Meaningful HTML

AMBIGUOUS STRUCTURE

(AKA "<DIV> SOUP")



IDENTIFIABLE SECTIONS

(AKA "SEMANTIC MARKUP")

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;
}
```

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

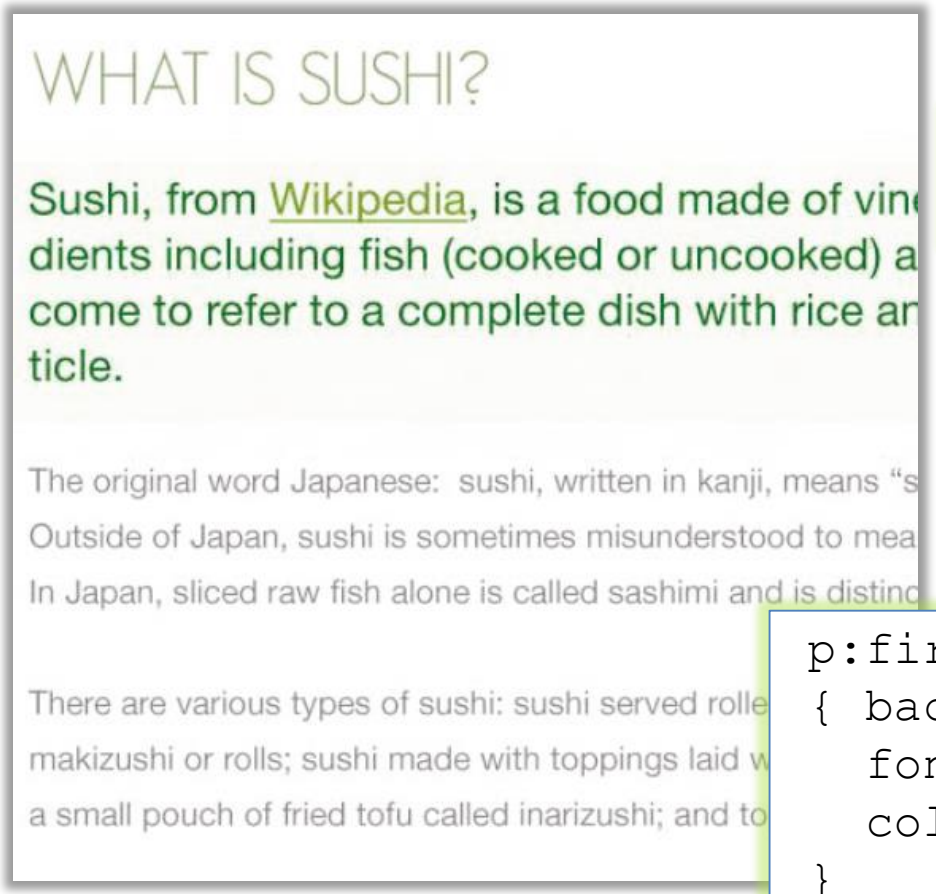
odd = 2n+1
even = 2n


```
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

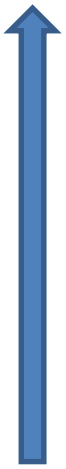
- Pseudo-elements
  - ::selection
    Targets elements that have been highlighted by the user
- Example: match any user-selected text within a textarea element

```
textarea::selection
```

CSS CASCADING

Cascading Style Sheets

- The term “cascading” means that a document can include more than one style sheet
- In this case, visualization follows priority rules
  - 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

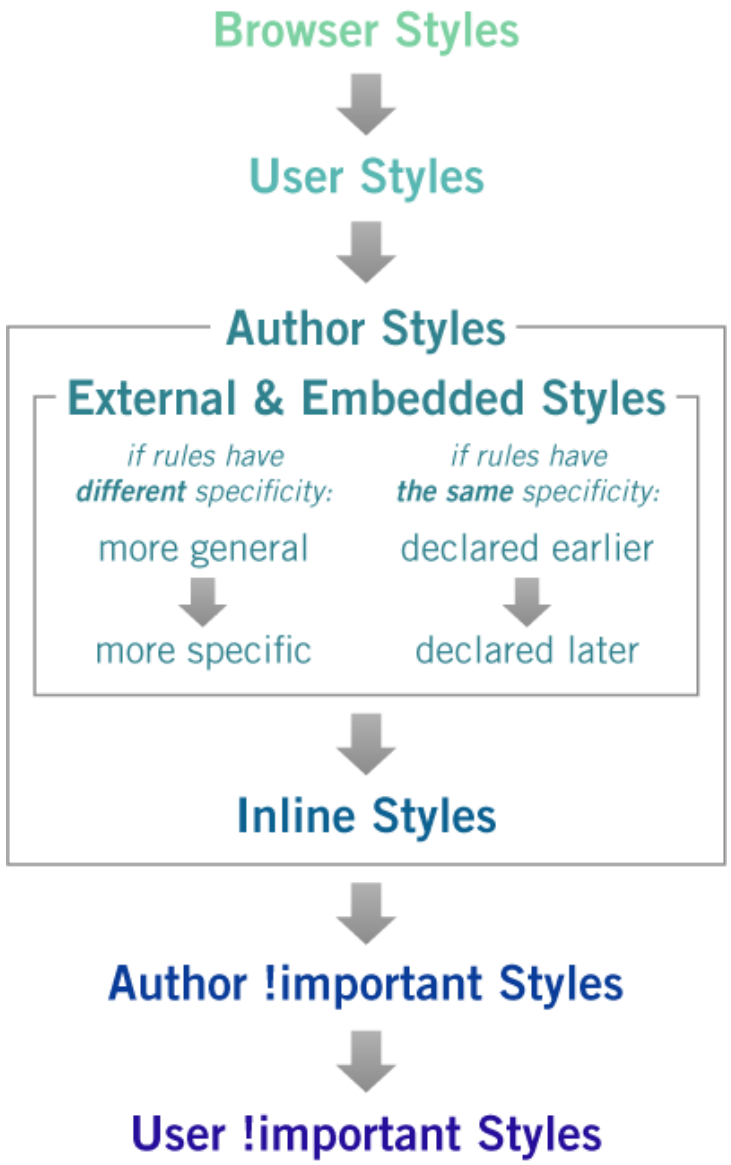
- <style> attribute within an HTML element

```
<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 specificity

- Specificity determines which CSS rule is applied by the browser
  - Specificity is usually the reason why your CSS-rules don't apply to some elements, although you think they should
- Every selector has its place in the specificity hierarchy
- There are four distinct categories which define the specificity level of a given selector
  - Inline styles (presence of style in document: attached directly to the element to be styled, e.g. <h1 style="color: #fff;">
  - IDs (# of ID selectors)
  - Classes, attributes and pseudo-classes (# of class selectors) – this group includes .classes, [attributes] and pseudo-classes such as :hover, :focus, …
  - Elements and pseudo-elements (# of Element (type) selectors), including for instance :before and :after

CSS specificity measure

- Start at 0, add 1000 for style attribute, add 100 for each ID, add 10 for each attribute, class or pseudo-class, add 1 for each element name or pseudo-element
- Examples

```
body #content .data img:hover
```

  - specificity value: 122 (0,1,2,2 or 0122): 100 for #content, 10 for .data, 10 for :hover, 1 for body and 1 for img
  - https://www.smashingmagazine.com/2007/07/css-specificity-things-you-should-know/

https://stuffandnonsense.co.uk/archives/css_specificity_wars.html

element selector

Specificity: 0,0,1

a

1 x element selector

Sith power: 0,0,1

p a

2 x element selectors

Sith power: 0,0,2

.foo

1 x class selector *

Sith power: 0,1,0

a.foo

1 x element selector
1 x class selector

Sith power: 0,1,1

p a.foo

2 x element selectors
1 x class selector

Sith power: 0,1,2

.foo .bar

2 x class selectors

Sith power: 0,2,0

p.foo a.bar

2 x element selectors
2 x class selectors

Sith power: 0,2,2

#foo

1 x id selector

Sith power: 1,0,0

a#foo

1 x element selector
1 x id selector

Sith power: 1,0,1

.foo a#bar

1 x element selector
1 x class selector
1 x id selector

Sith power: 1,1,1

.foo .foo #foo

2 x class selectors
1 x id selector

Sith power: 1,2,0

style

1 x style attribute

Sith power: 1,0,0,0

CSS specificity principles

- The embedded style sheet has a greater specificity than other rules
- ID selectors have a higher specificity than attribute selectors
  - You should always try to use IDs to increase the specificity
- A class selector beats any number of element selectors
- The universal selector and inherited selectors have a specificity of 0, 0, 0, 0

CSS specificity principles

- If two selectors apply to the same element, the one with higher specificity wins

- Equal specificity
  - The latest rule is the one that counts
  - The last rule defined overrides any previous, conflicting rule

```
#content h1 { padding: 5px; }
#content h1 { padding: 10px; }
```

  - Both rules have the specificity 0, 1, 0, 1: the second rule is applied

CSS specificity example

```
A:
h1 { color: blue; }

B:
#content h1 { color: green; }

C:
<div id="content">
  <h1 style="color: #fff">Headline</h1>
</div>
```

0,0,0,1

0,1,0,1

1,0,0,0

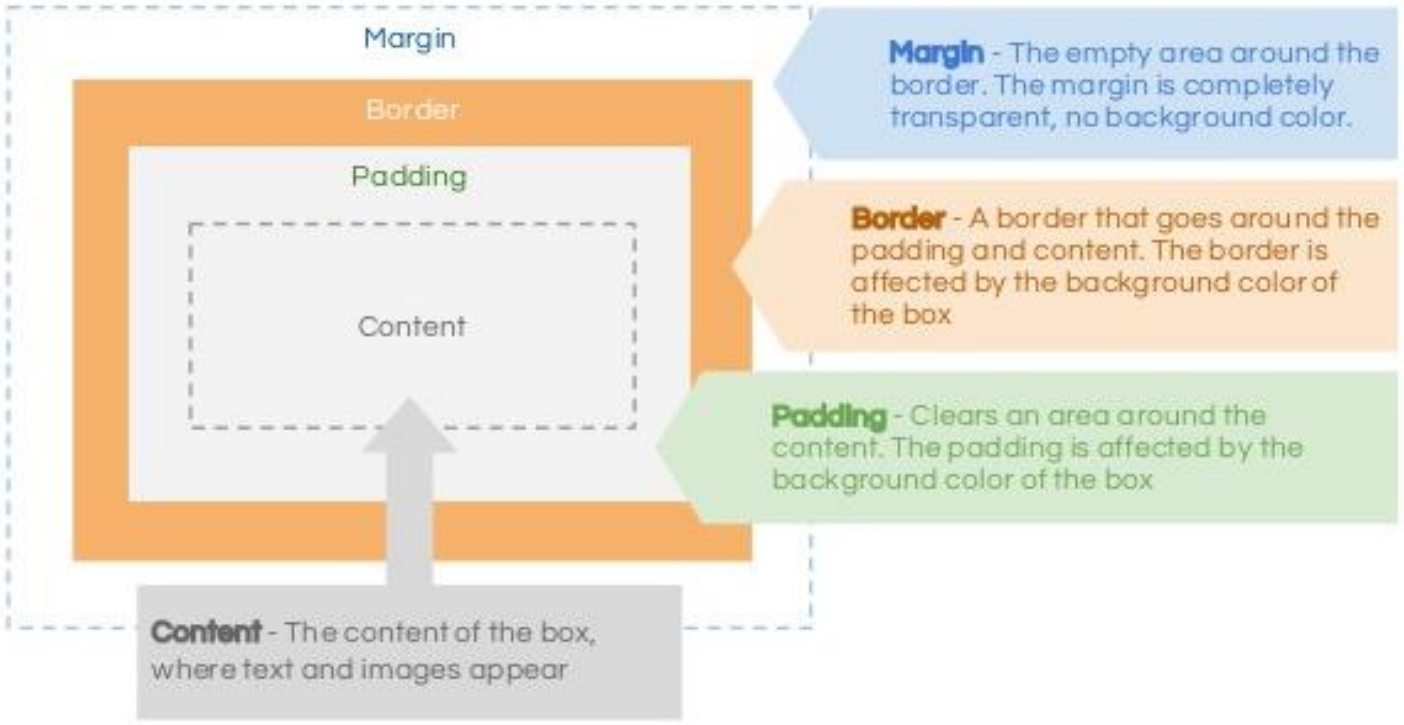
References for CSS selectors

- Selectutorial
  - http://css.maxdesign.com.au/selectutorial/index.htm
- CSS selectors
  - https://www.sitepoint.com/css-selectors/
- CSS Specificity: Things You Should Know
  - https://www.smashingmagazine.com/2007/07/css-specificity-things-you-should-know/

CSS BOX MODEL

The box model

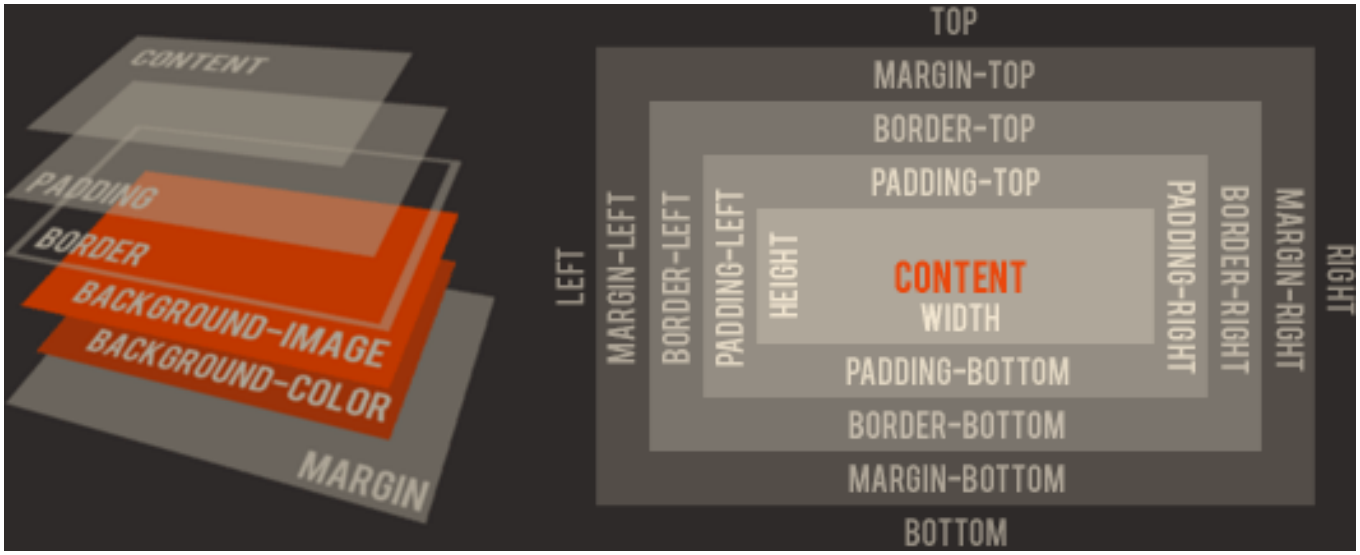
- One of the cornerstones of CSS
- Every element on the page is considered to be a rectangular box



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

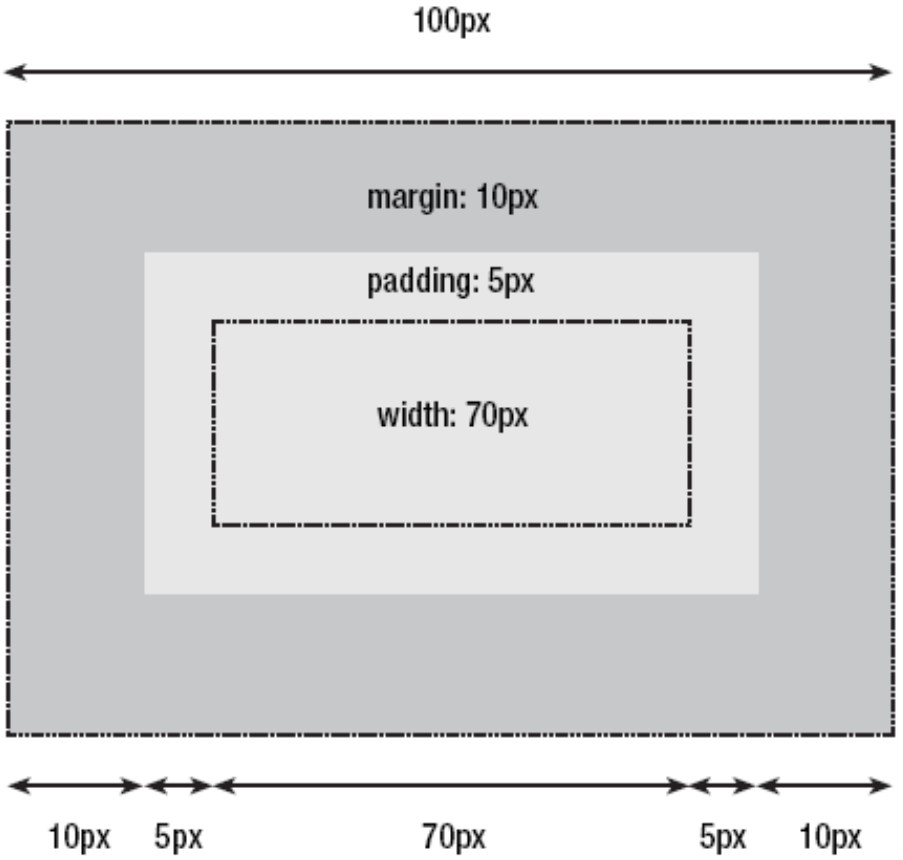


- You can set any of these properties, independently

Example

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

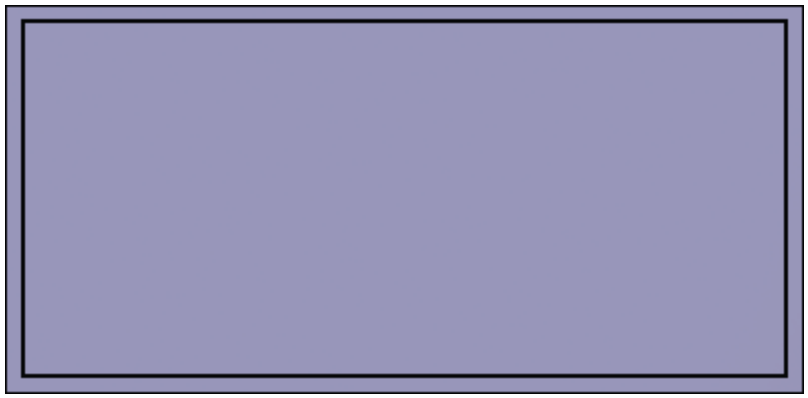
```
#myBox {
  margin: 10px;
  padding: 5px;
  width: 70px;
}
```



Box and borders

- Border-color
  - Allows for multiple border colors to be specified, one pixel at a time

```
border: 5px solid #000;
border-color: #000 transparent transparent #000;
```



Box and borders

- Border-radius
  - Curves the corners of the border using the radius given, usually in pixels
  - Can be given to all corners, or only to individual corners as specified

```
border-radius: 25px;
```

```
border-top-right-radius: 25px;
```

Box and borders

top-right & bottom-left

```
border-radius: 40px 25px;
```

top-left & bottom-right

top-right & bottom-left

```
border-radius: 40px 20px 0;
```

top-left bottom-right

top-right bottom-right

```
border-radius: 40px 25px 0 50px;
```

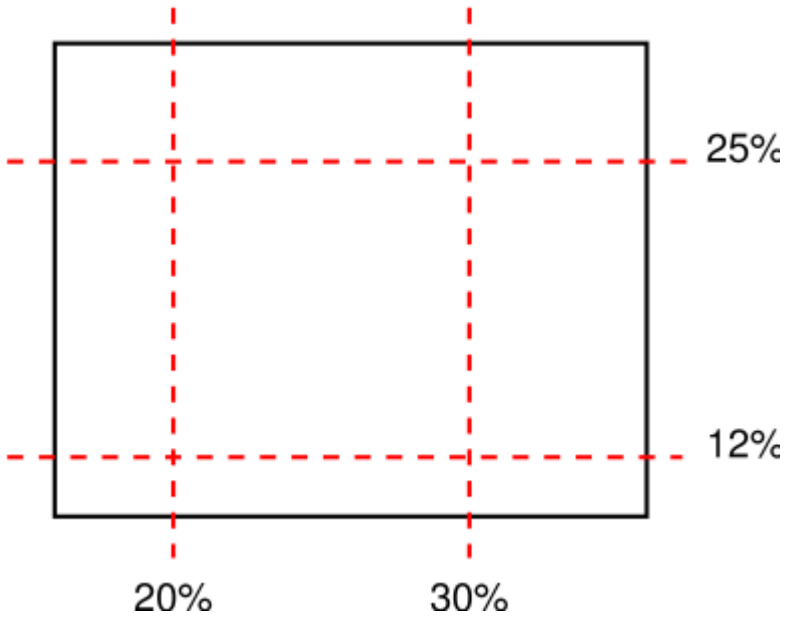
top-left bottom-left

Box and borders

- Border-image
  - Possibility to use an image in place of the border styles
  - In this case, the border design is taken from the sides and corners of a specified image, whose pieces may be sliced, scaled and stretched in various ways to fit the size of the border image area
  - The border-image properties do not affect layout: layout of the box, its content, and surrounding content is based on the ‘border-width’ and ‘border-style’ properties only

Border image

- *'stretch'*
  - The image is stretched to fill the area
- *'repeat'*
  - The image is tiled (repeated) to fill the area
- *'round'*
  - The image is tiled (repeated) to fill the area
  - If it does not fill the area with a whole number of tiles, the image is rescaled so that it does



where to slice (9 parts)

```
border-image: url(border.png) 25% 30% 12% 20% repeat;
```

image source

how to apply

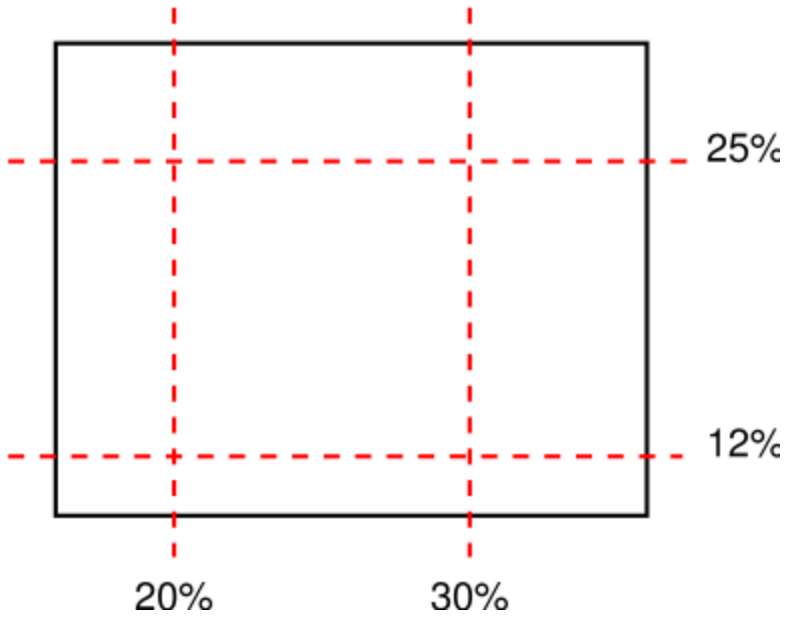
Border image

where to slice (9 parts)

```
border-image: url(border.png) 25% 30% 12% 20% repeat;
```

image source

how to apply

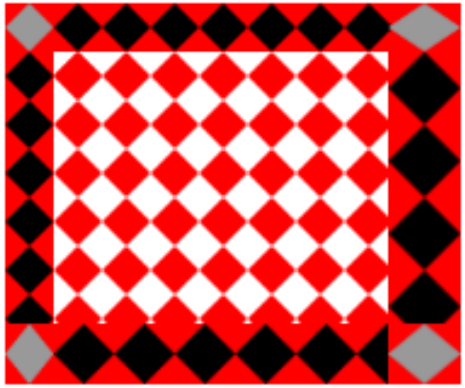


- *'stretch'*
  - The image is stretched to fill the area
- *'repeat'*
  - The image is tiled (repeated) to fill the area
- *'round'*
  - The image is tiled (repeated) to fill the area
  - If it does not fill the area with a whole number of tiles, the image is rescaled so that it does

Example

diamond.png

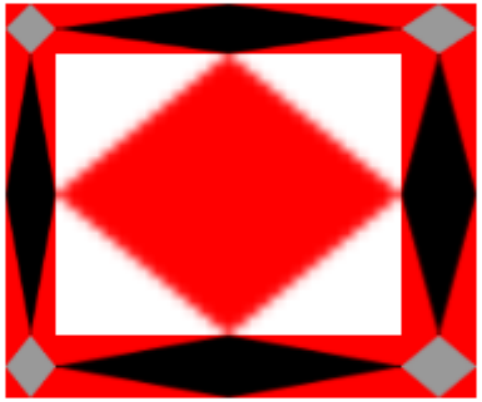
```
div#demo
{ border: solid transparent;
  border-width: 20px 30px 25px 20px;
  border-image: url("diamonds.png") 33% repeat }
```

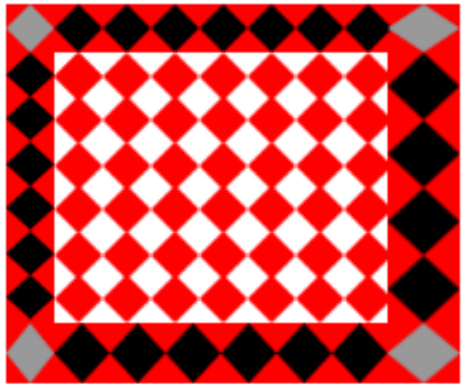


Example

diamond.png

```
div#demo
{ border: solid transparent;
  border-width: 20px 30px 25px 20px;
  border-image: url("diamonds.png") 33% round }
```

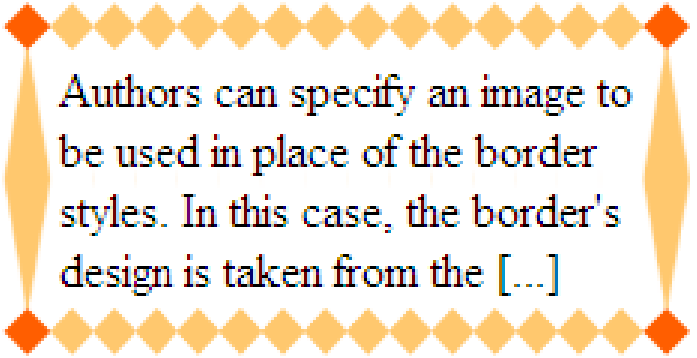




```
div#demo
{ border: solid transparent;
  border-width: 20px 30px 25px 20px;
  border-image: url("diamonds.png") 33% stretch }
```

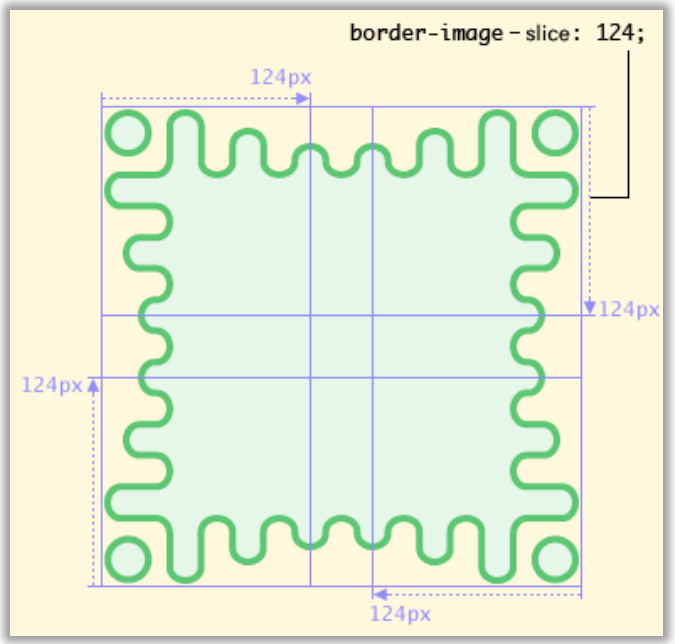
Example

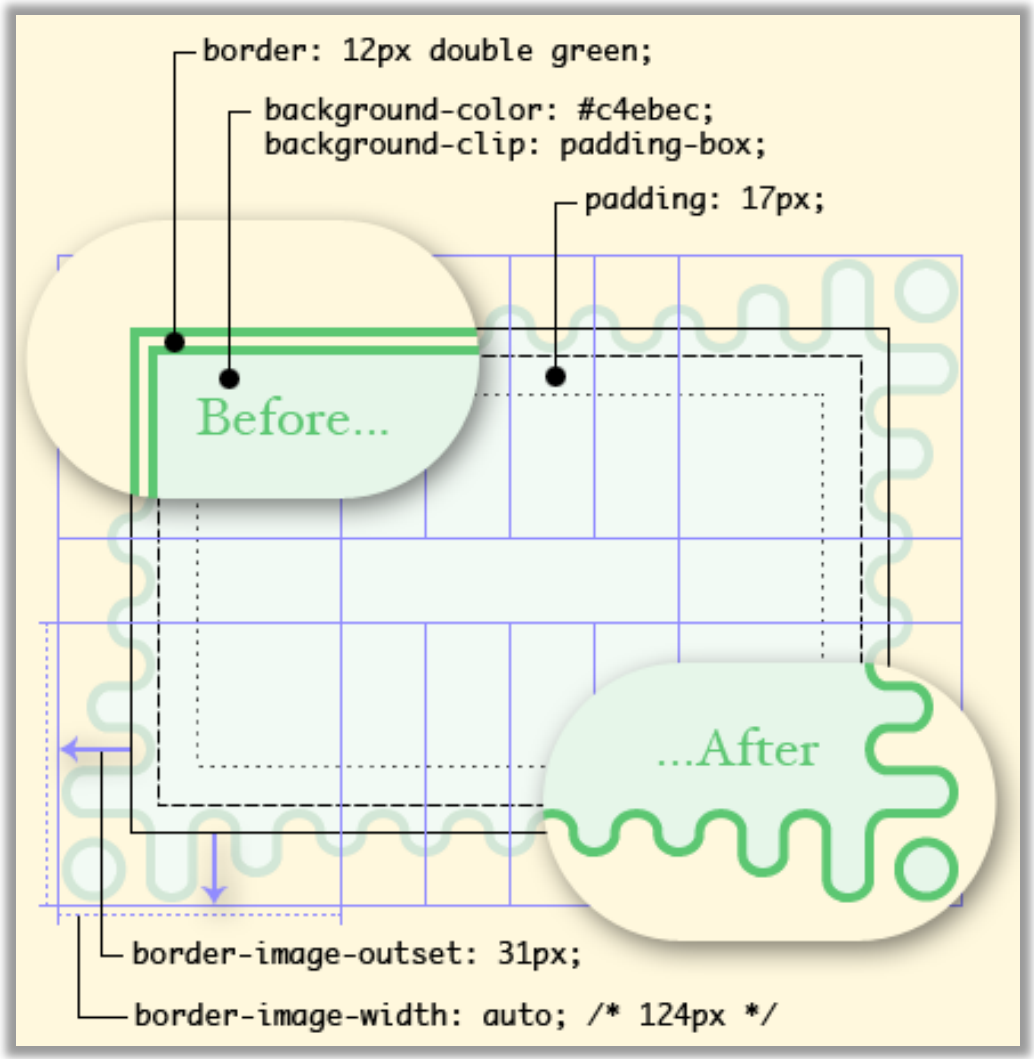
border.png
(81x81 px)




```
p
{ width: 12em;
  height: 5em;
  margin: 5em;
  padding: 3px;
  border: double orange 1em;
  border-image: url("border.png") 27 27 27 27 round stretch; }
```

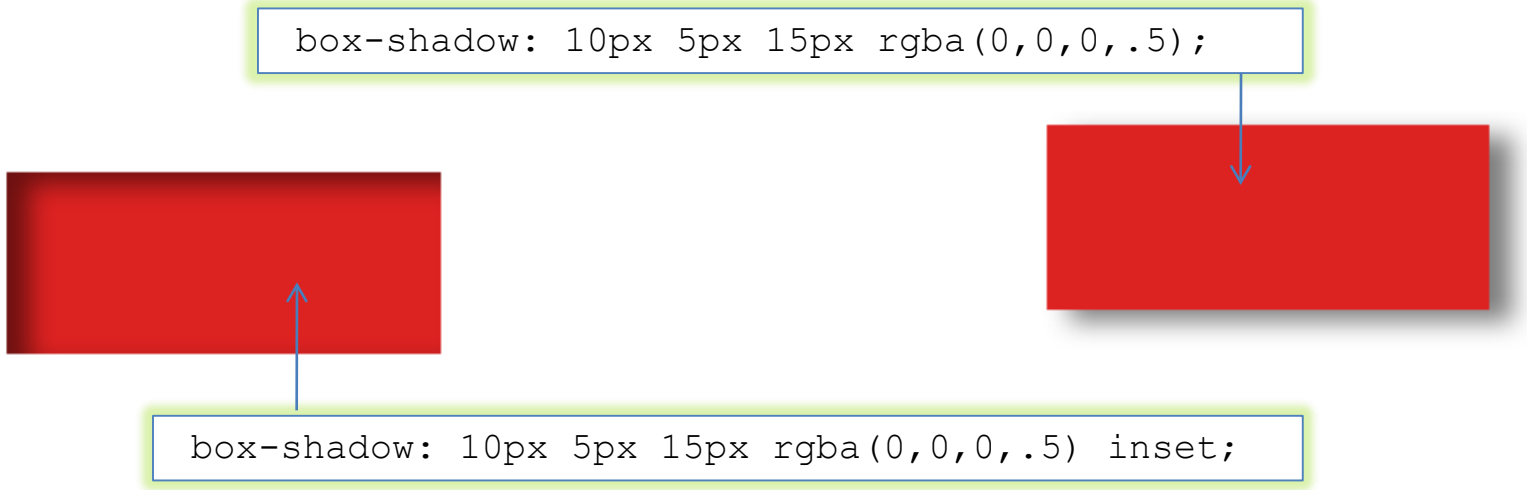
Example


border-image-slice: 124;
124px
124px
124px
124px


border: 12px double green;
background-color: #c4ebec;
background-clip: padding-box;
padding: 17px;
Before...
...After
border-image-outset: 31px;
border-image-width: auto; /* 124px */

Box shadow

- Box-shadow
  - Creates a drop shadow beneath the selected element
  - The first argument is the horizontal offset, the second is the vertical offset, the third is the blur radius, and the final argument is the color to be used as the shadow



Box shadow

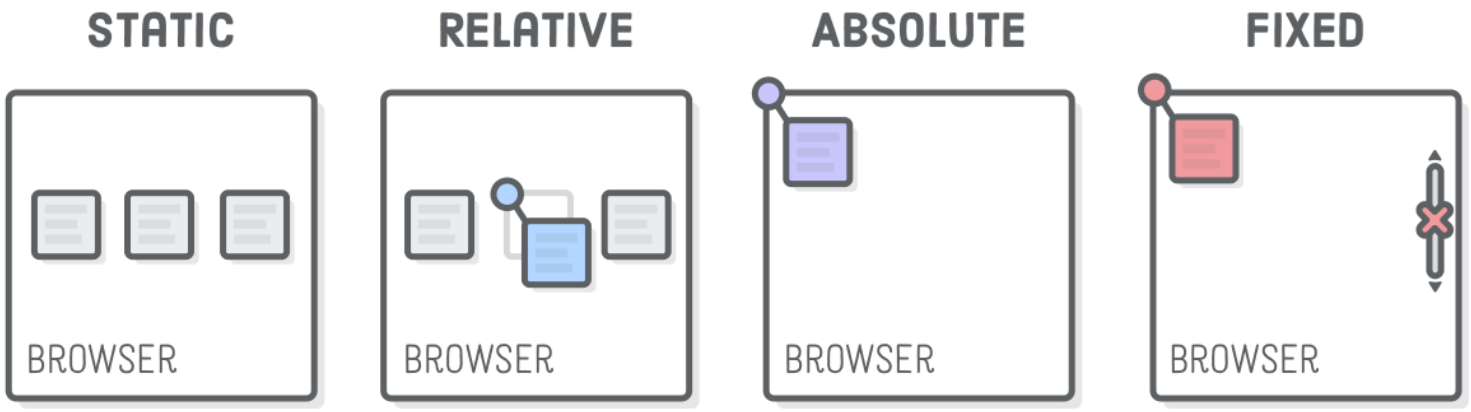
- Multiple shadows

```
#Example_M {
  -moz-box-shadow: 0 0 5px black, 40px -30px lime,40px 30px
    50px red, -40px 30px yellow, -40px -30px 50px blue;
  -webkit-box-shadow: 0 0 5px black, 40px -30px lime, 40px
    30px 50px red, -40px 30px yellow, -40px -30px 50px blue;
  box-shadow: 0 0 5px black, 40px -30px lime, 40px 30px
    50px red, -40px 30px yellow, -40px -30px 50px blue;
}
```

CSS POSITIONING SCHEMES

Positioning schemes

- In CSS there are four basic 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)

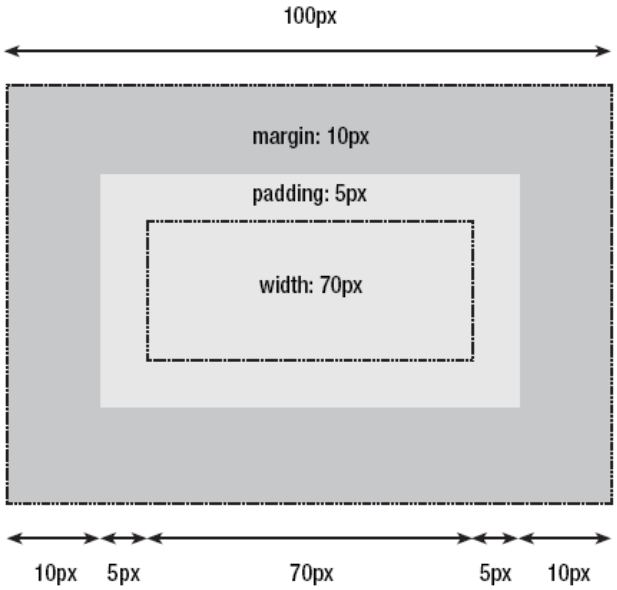


https://internetingishard.com/html-and-css/advanced-positioning/

Normal flow

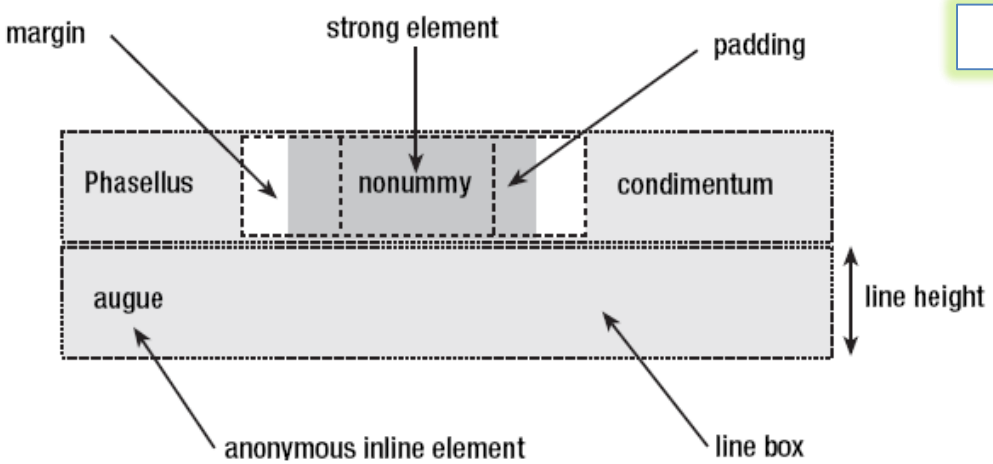
- Unless specified, all boxes start life being positioned in the normal flow
  - The position of an element's box in the normal flow is dictated by that element's position in the HTML code
- Block-level boxes will appear vertically one after the other
  - The vertical distance between boxes is calculated by the boxes' vertical margins

```
<div> … </div>
```



Normal flow

- Inline boxes are laid out in a line horizontally
  - 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

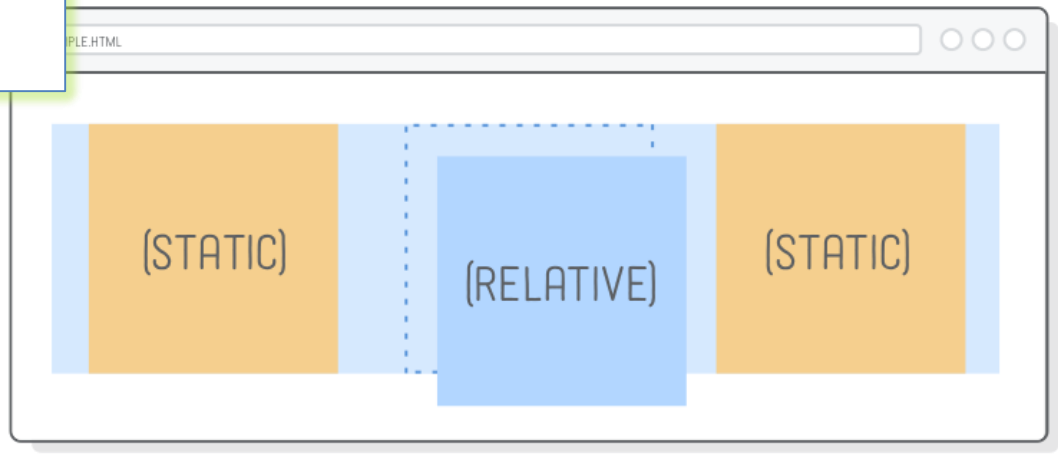



```
<span> … </span>
```

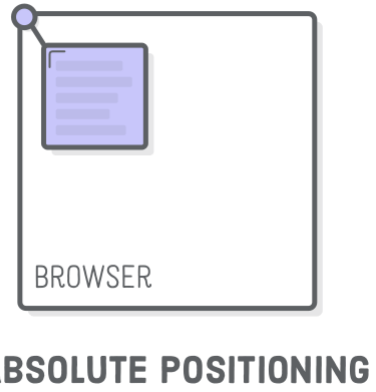
Relative positioning

- One element can be shifted “relative” to its normal flow position by setting a vertical and/or horizontal offset

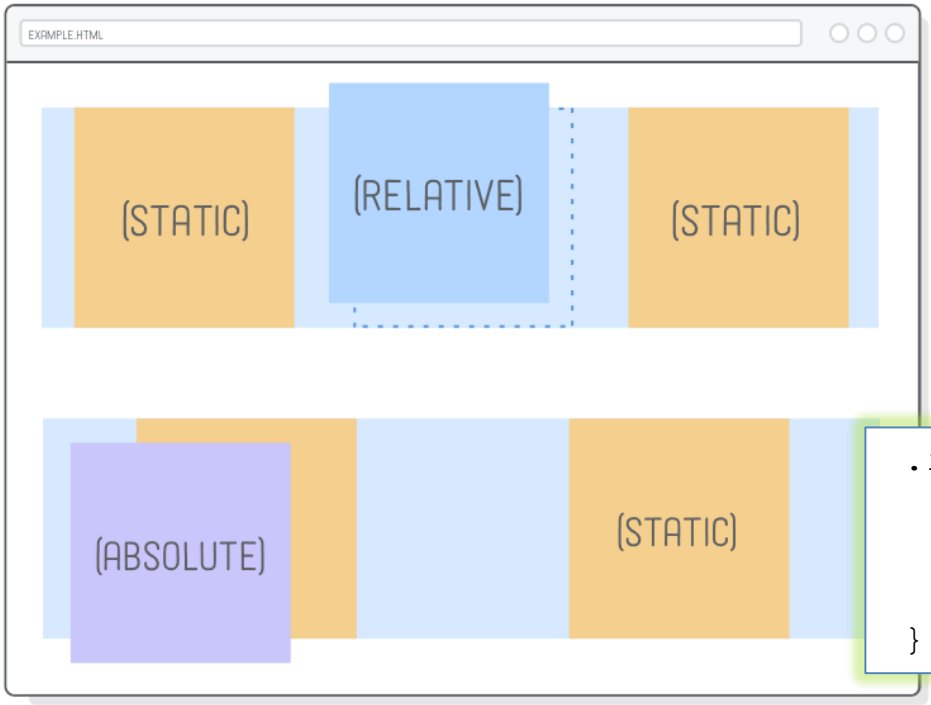
```
.item-relative {
  position: relative;
  left: 20px;
  top: 20px;
}
```



Absolute positioning



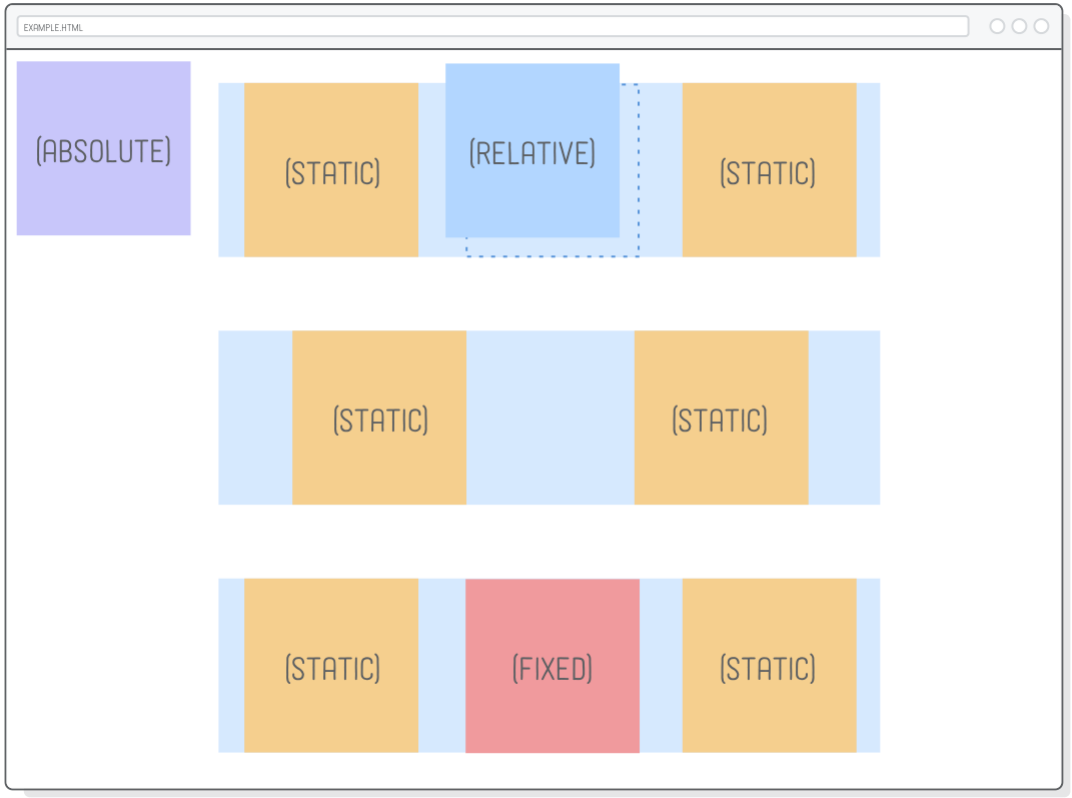
- Takes the element out of the flow of the document, thus taking up no space
- Other elements in the normal flow of the document will act as though the absolutely positioned element was never there




```
.item-absolute {
  position: absolute;
  top: 30px;
  left: 350px;
}
```

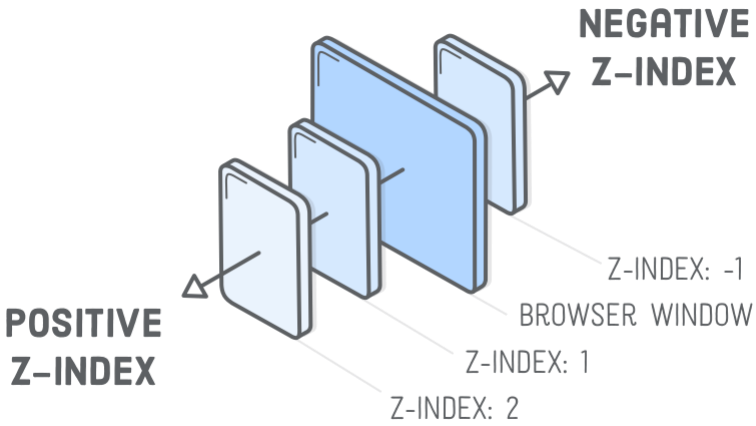
Fixed positioning

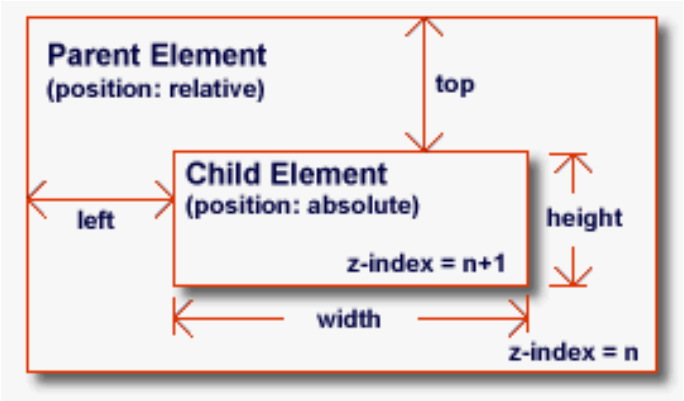
- Has a lot in common with absolute positioning: the element is removed from the normal flow of the page, and the coordinate system is relative to the entire browser window
- The key difference is that fixed elements don't scroll with the rest of the page



Fixed positioning and z-index

- 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)

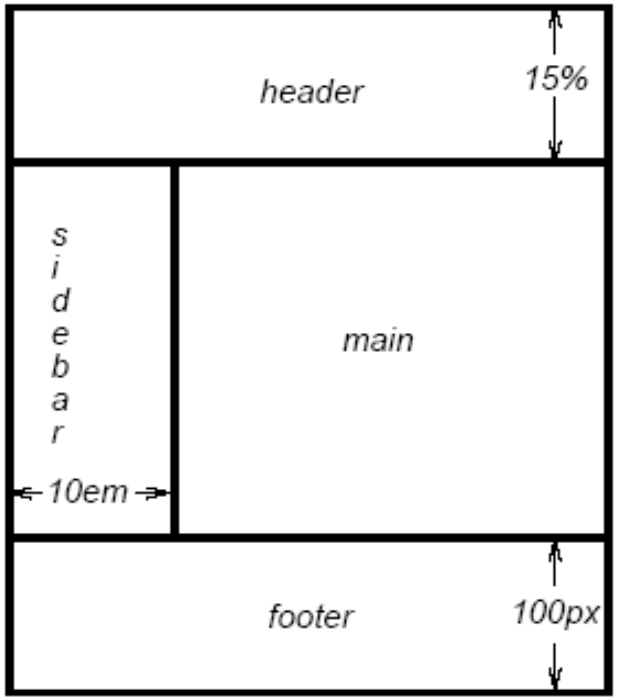




Fixed positioning

- 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

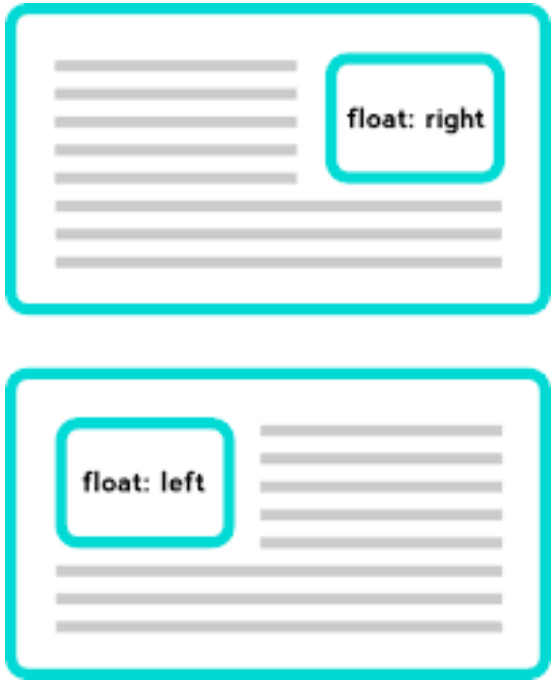
- The CSS float property gives control over the horizontal position of an element
- Tools necessary to align block-level elements: floats for left/right alignment and auto-margins for center alignment
  - This only applies to block boxes
  - Inline boxes are aligned with the text-align property

https://internetingishard.com/html-and-css/floats/

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

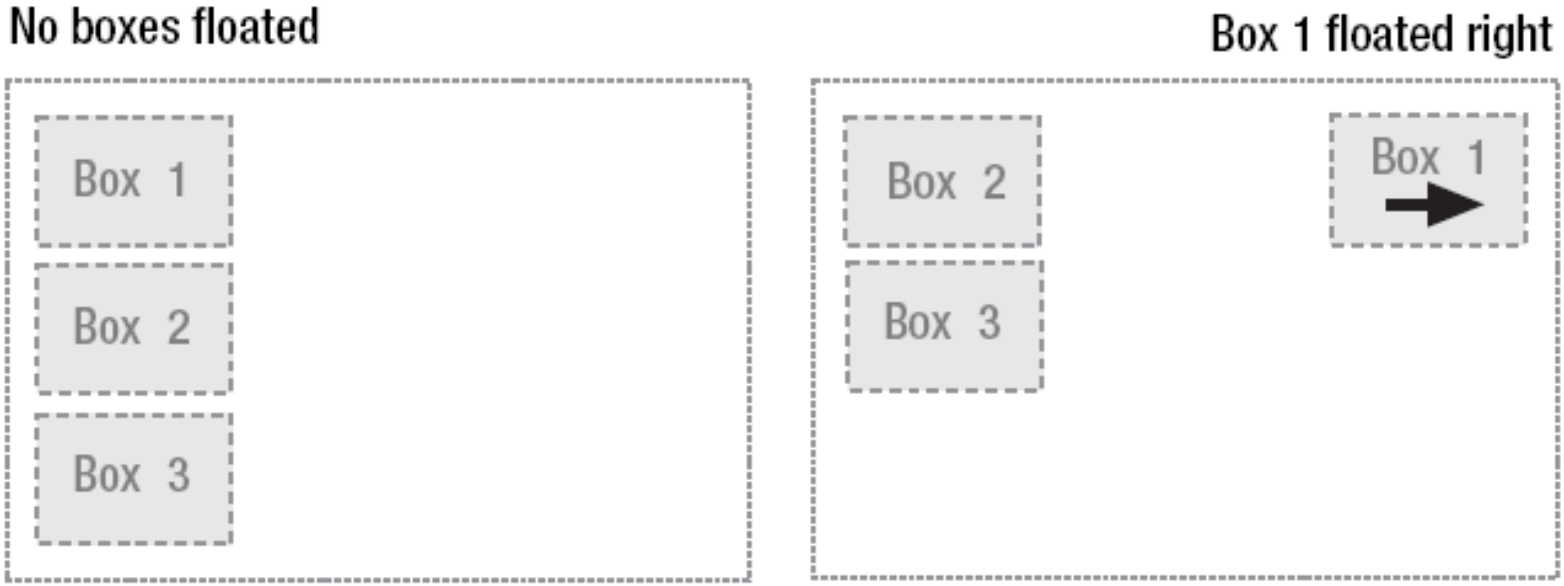
```
img
{
   float:right;
}
```



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

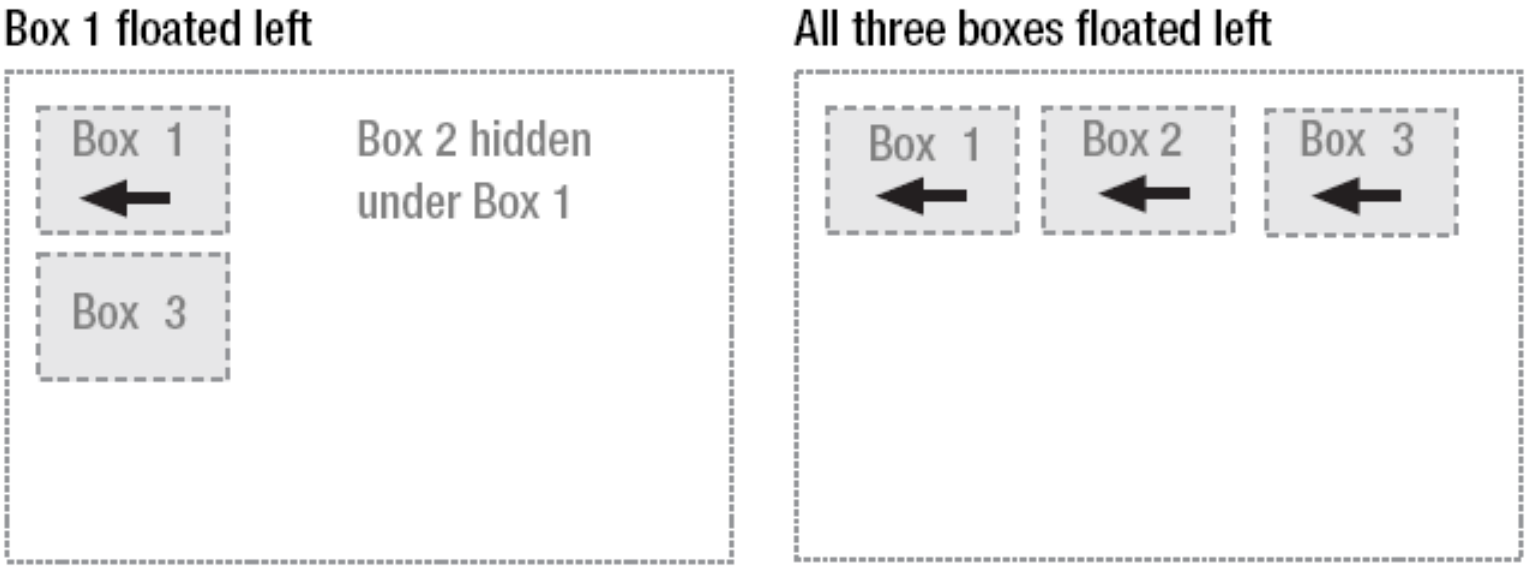
Floating

- 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



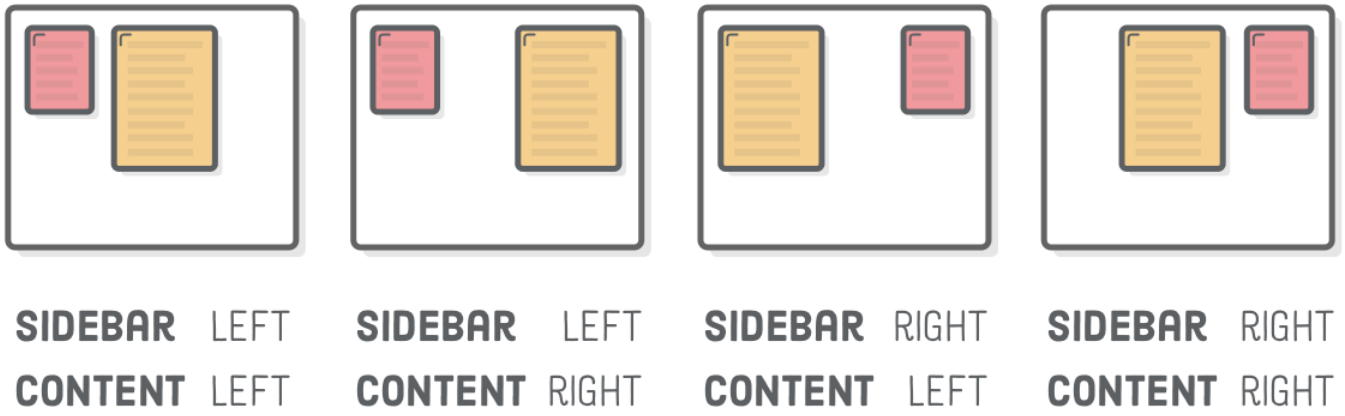
Floating

- 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



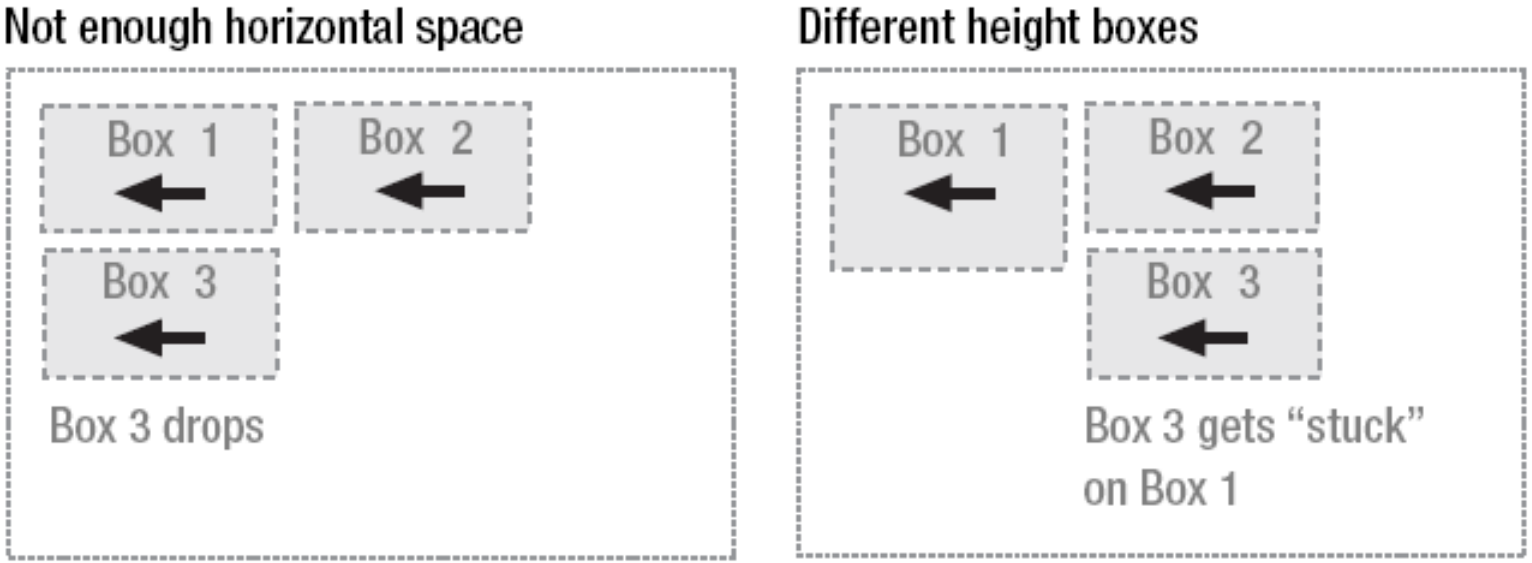
Floating

- When you float multiple elements in the same direction, they'll stack horizontally, much like the default vertical layout algorithm, except rotated 90 degrees



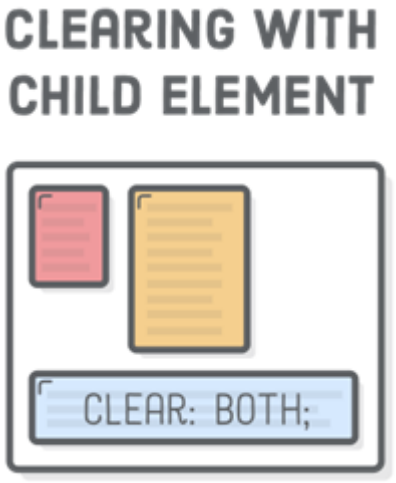
Floating

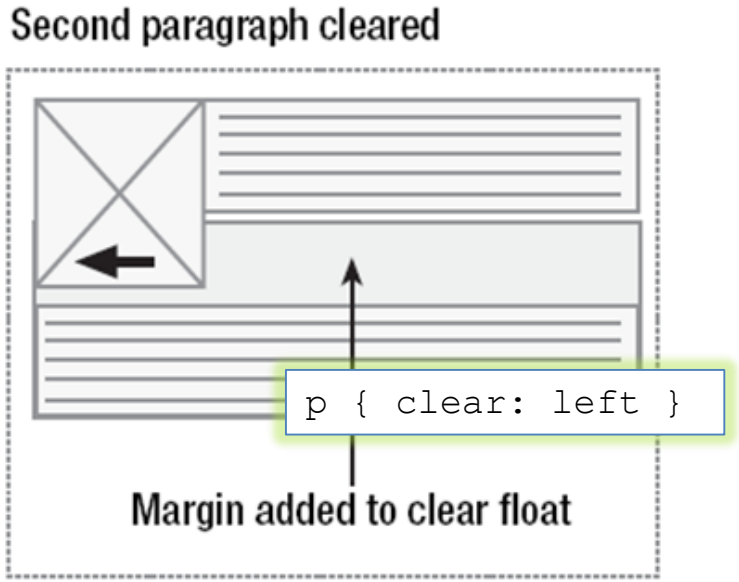
- If the containing block is too narrow for all of the floated elements to fit horizontally
  - The remaining floats will drop down until there is sufficient space
  - If the floated elements have different heights, it is possible for floats to get “stuck” on other



Clearing floats

- "Clearing" a float is when you tell a block to ignore any floats that appear before it
  - Instead of flowing around the floated box, a cleared element always appears after any floats
  - It's like forcing a box back into the default vertical flow of the page





References for CSS box model and positioning

- Learn CSS layout
  - http://learnlayout.com/
- Floatutorial
  - http://css.maxdesign.com.au/floatutorial/
- All about floats
  - https://css-tricks.com/all-about-floats/

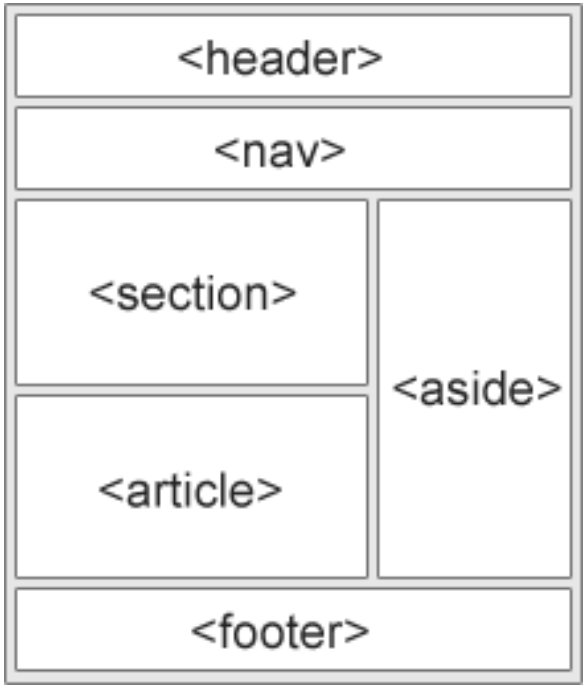
PAGE LAYOUT WITH CSS FLOATS

Page layout

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

- Example

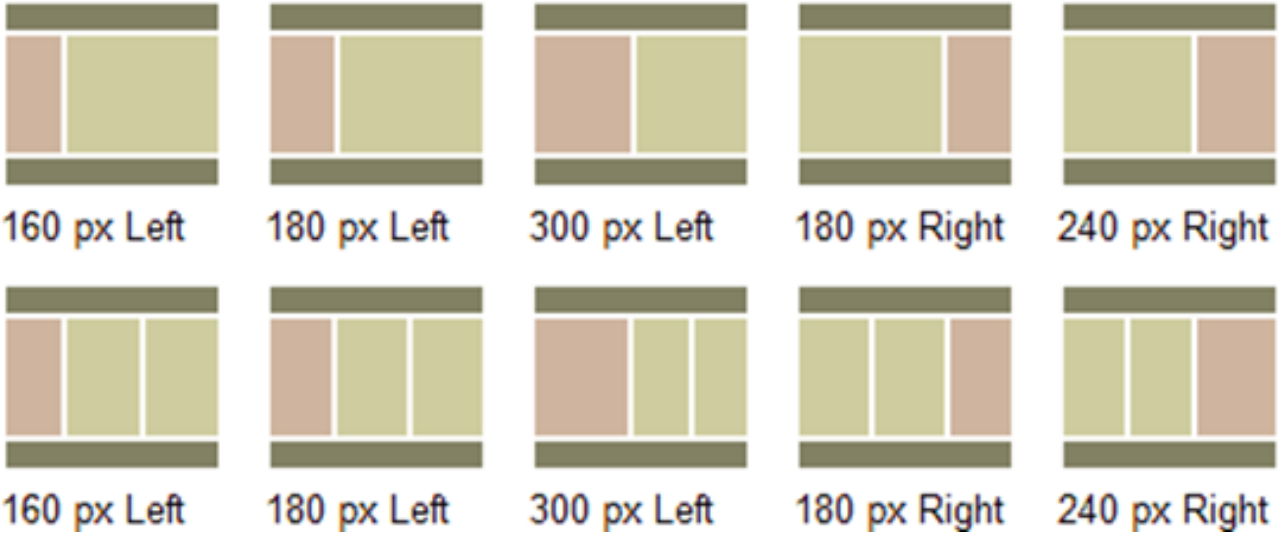
HTML5 semantic tags



- <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

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

- Auto margins

Auto margins

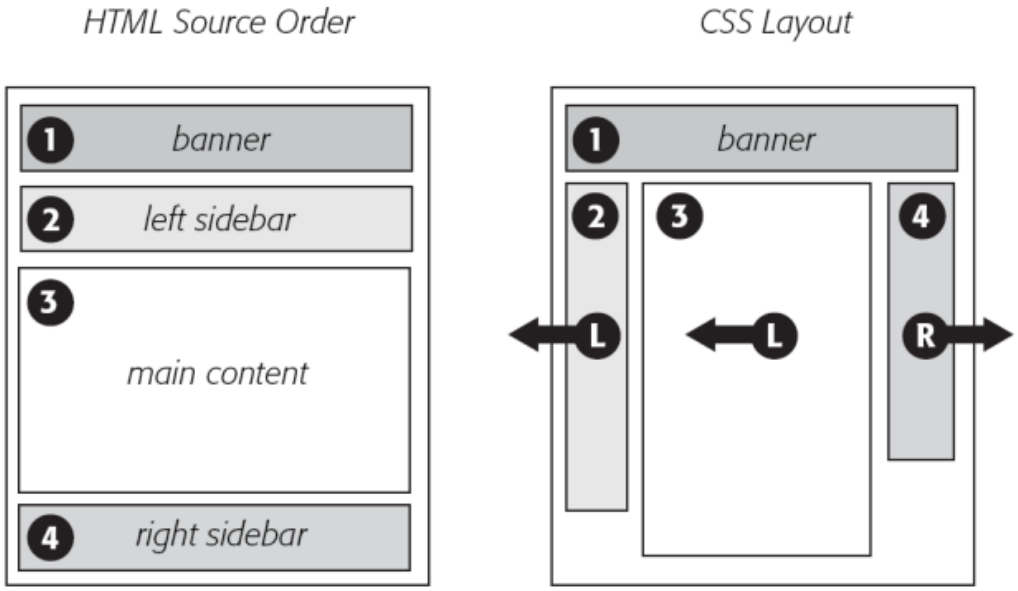
- When you set the left and right margin of a block-level element to auto, it will center the block in its parent element
  - Define the width of the wrapper div
  - Set the horizontal margin to auto

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

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

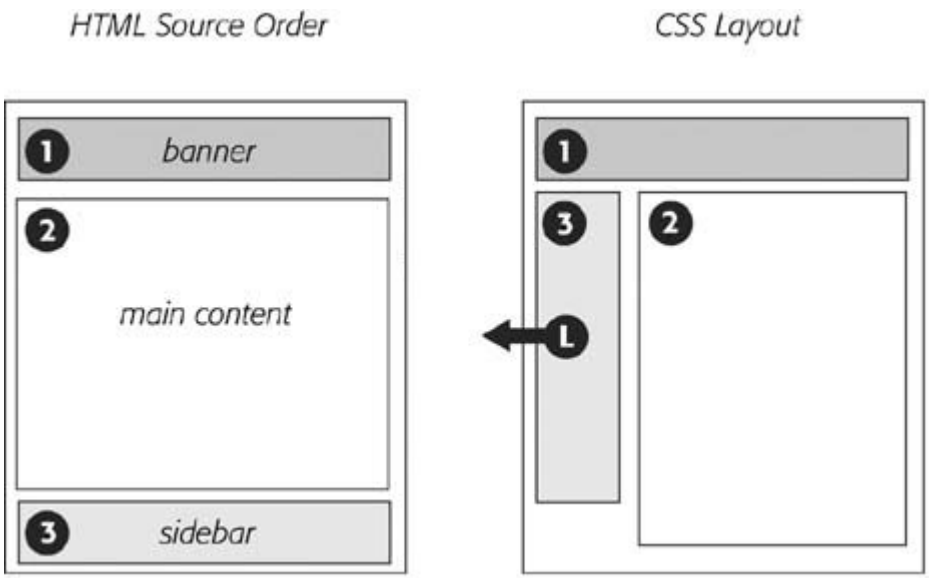
Float-based layouts

- 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

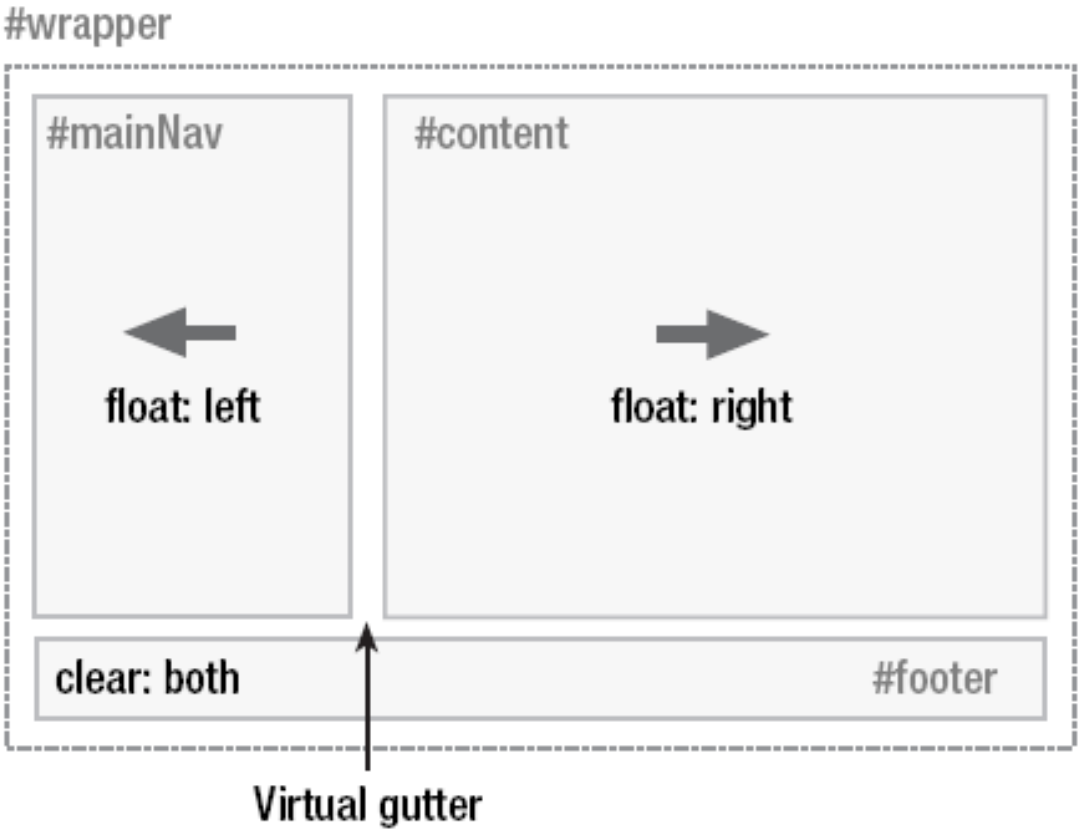
- HTML framework
  - Main navigation on the left side of the page
  - Content on the right
- For accessibility reasons the content area is above the navigation in the source
  - The main content is the most important element in the page and it so should come first in the document
  - There is no point 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

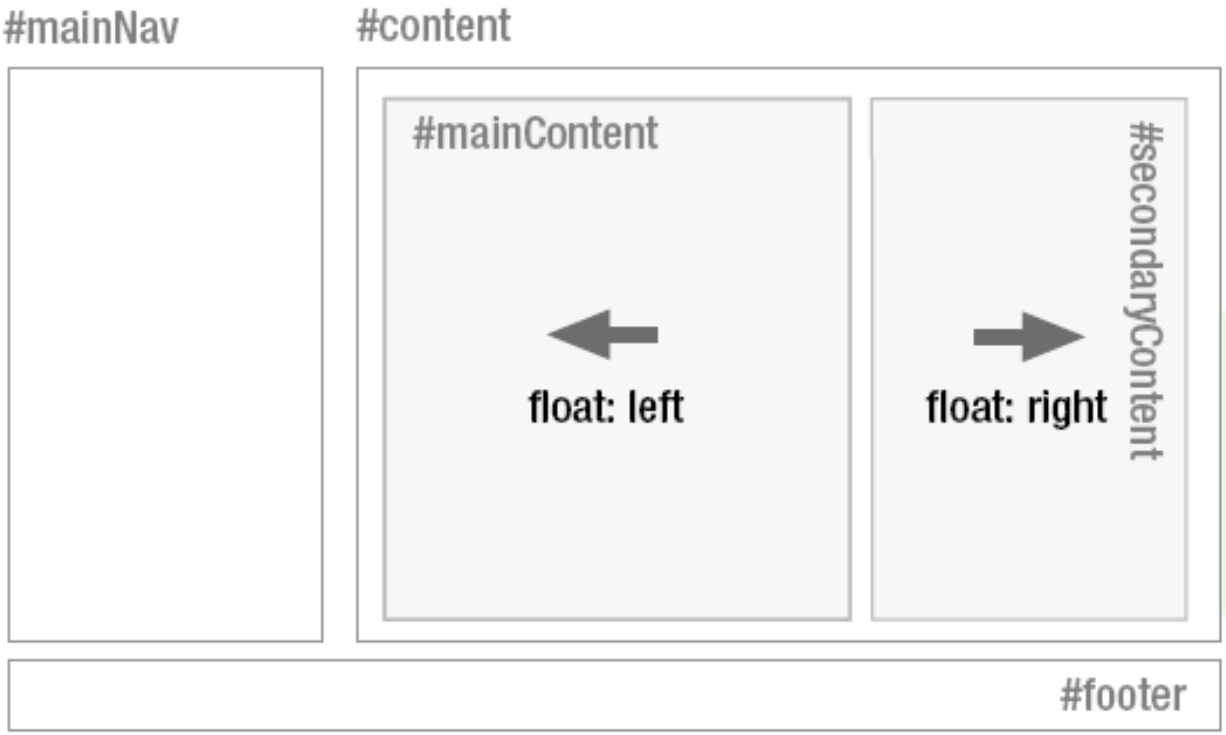
Three-column floated layout

- 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; }
```

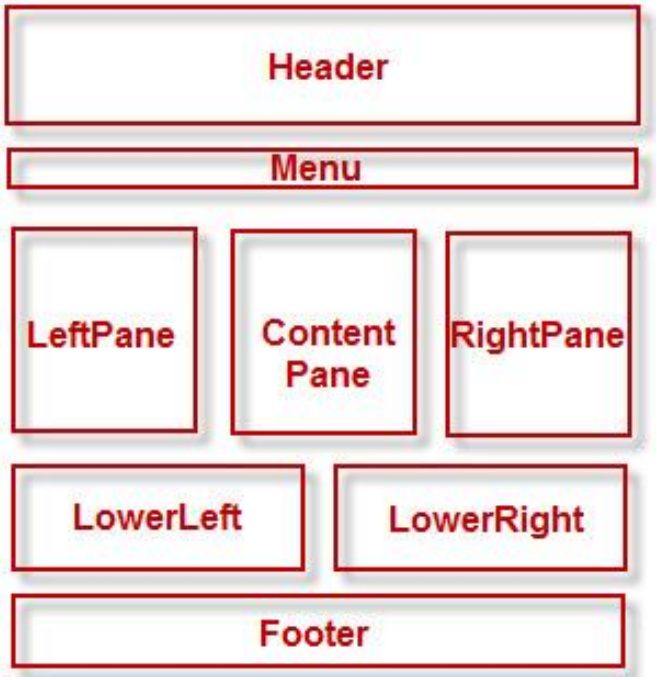
```
#secondaryContent h1, #secondaryContent h2,
     #secondaryContent p {
padding-left: 20px;
padding-right: 20px; }
```

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

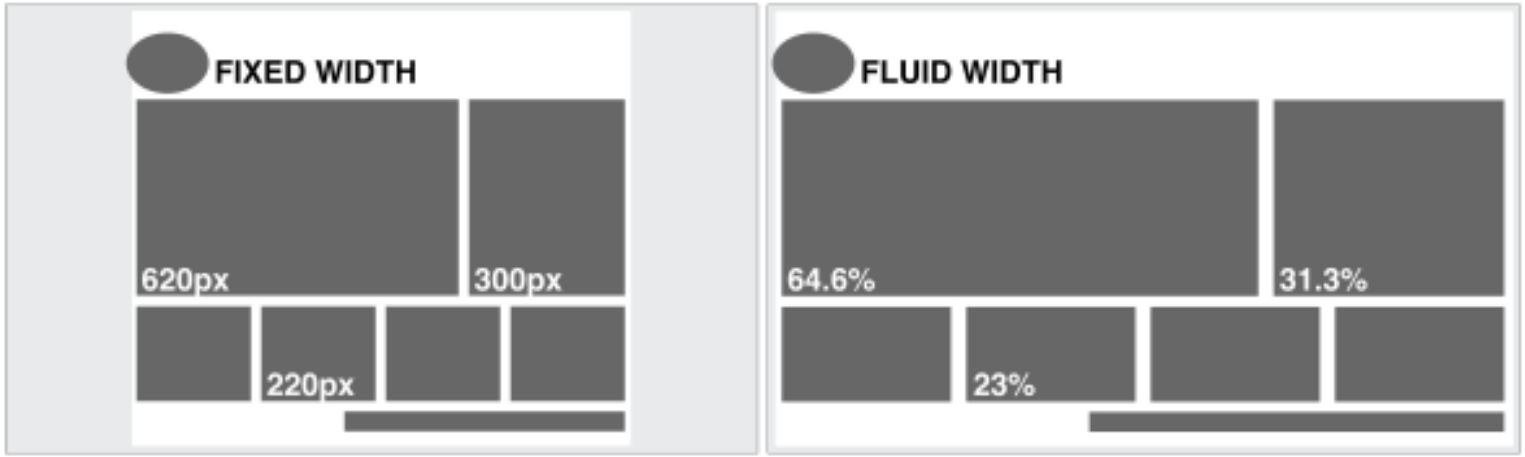



```
<div id="headerWrap">Header</div>
<div id="MenuWrap">Menu</div>
<div id="LeftPane">LeftPane</div>
<div id="ContentPane">ContentPane</div>
<div id="RightPane">RightPane</div>
<div id="LowerLeftPane">LowerLeft</div>
<div id="LowerRightPane">LowerRight</div>
<div id="footerWrap">Footer</div>
```

```
#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%; }
```

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

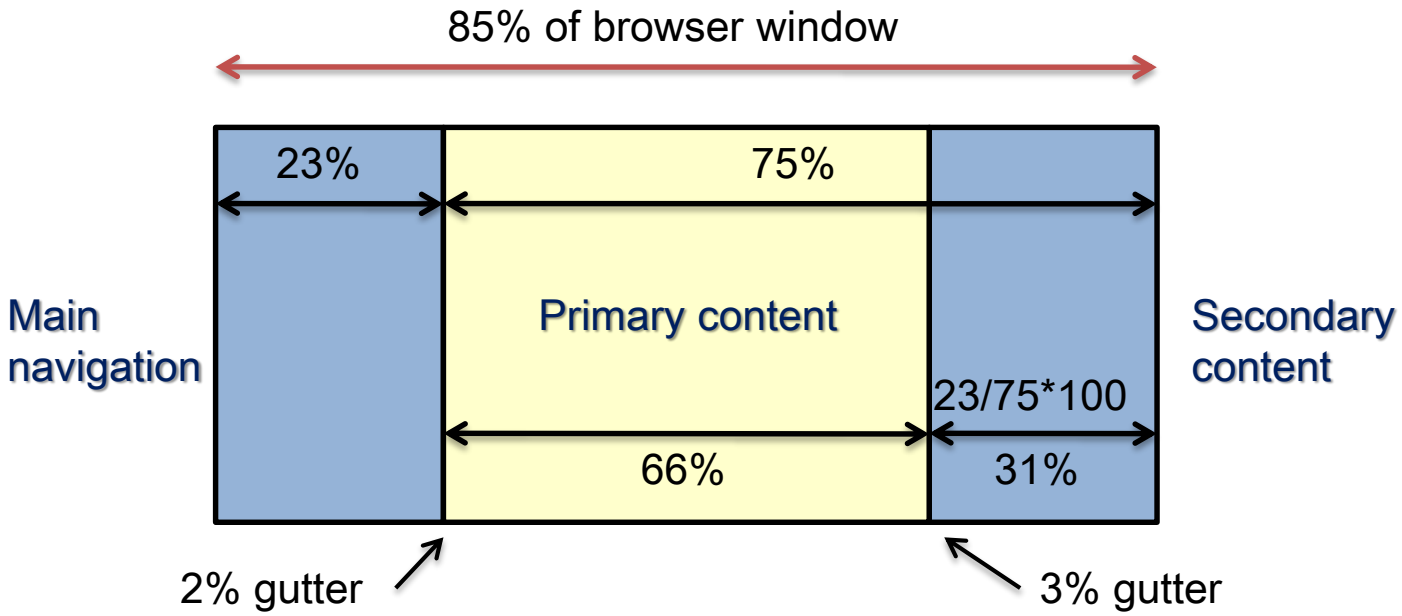
```
#wrapper {
width: 85%;
}
```

- 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;
}
```

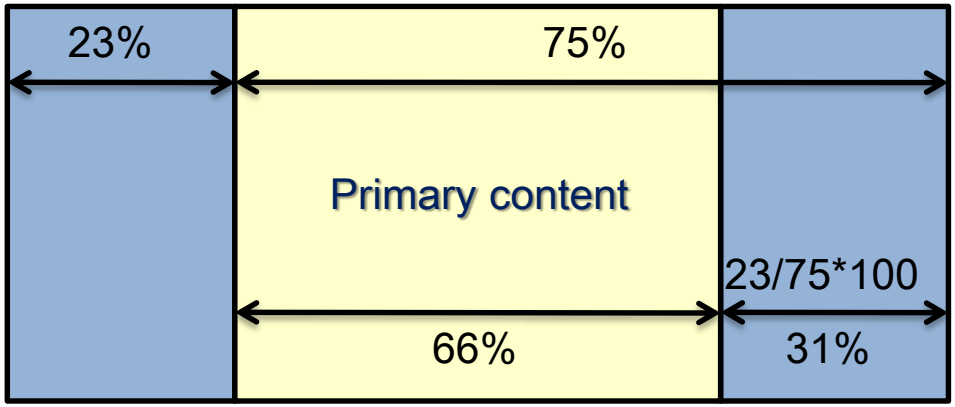
Liquid layout

- 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;
}
```

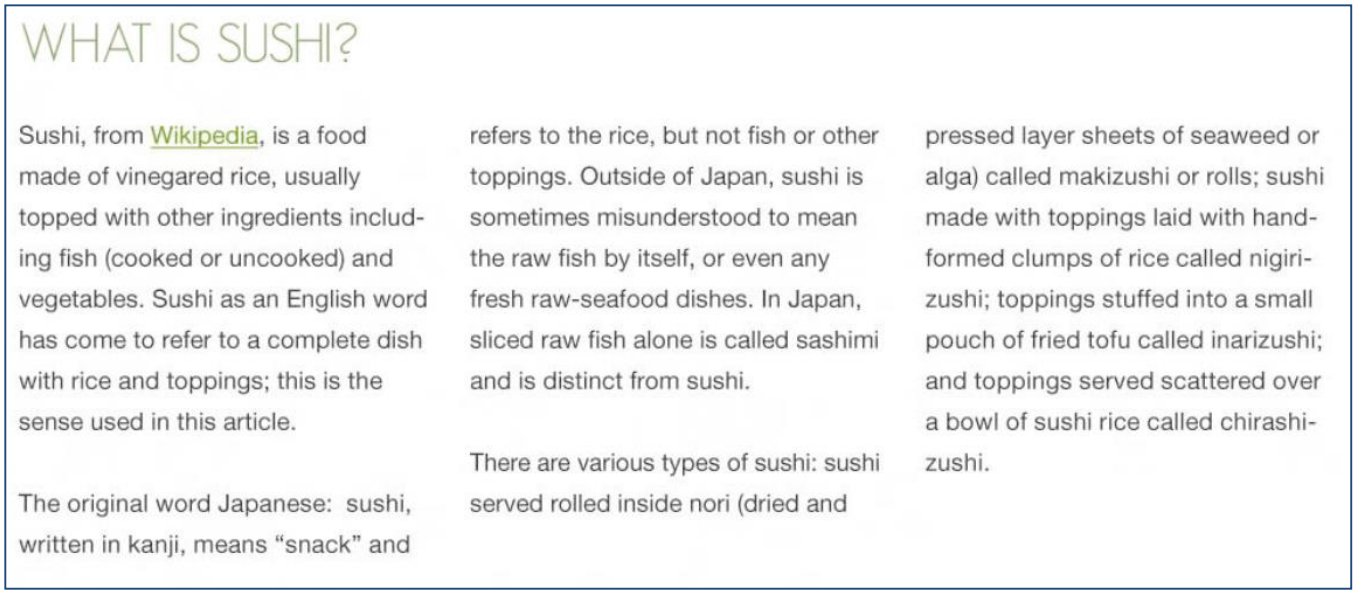
```
#content {
width: 52em;
max-width: 75%;
float: right;
}
#mainContent {
width: 32em;
max-width: 66%;
float: left;
}
#secondaryContent {
width: 18em;
max-width: 31%;
float: right;
}
```

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

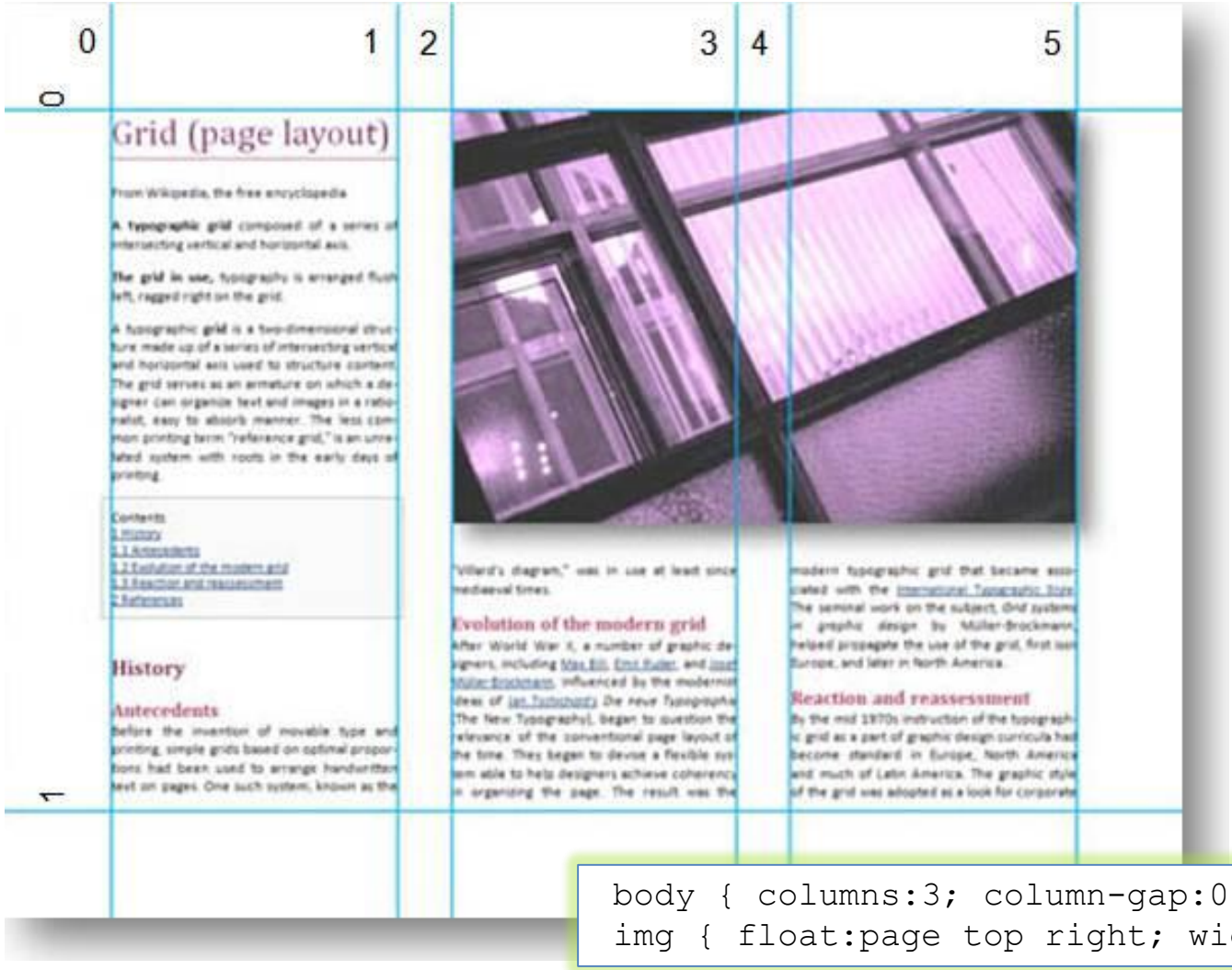
```
div
{ grid-columns:50% * * 200px; }
```

- 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); }
```

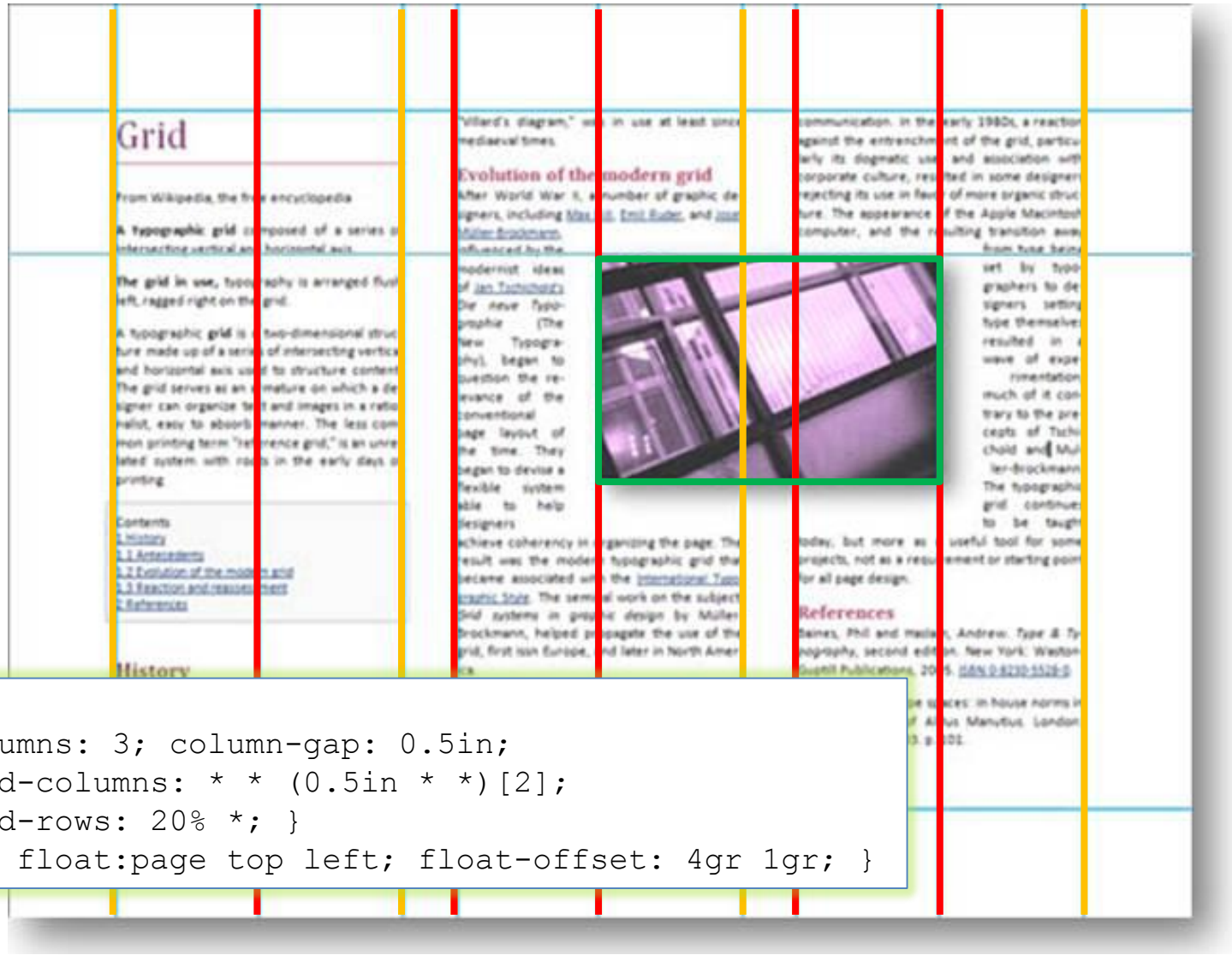
- Define a header row of 100 pixels, and add as many additional rows as necessary, alternating heights of 30 and 60 pixels

Example



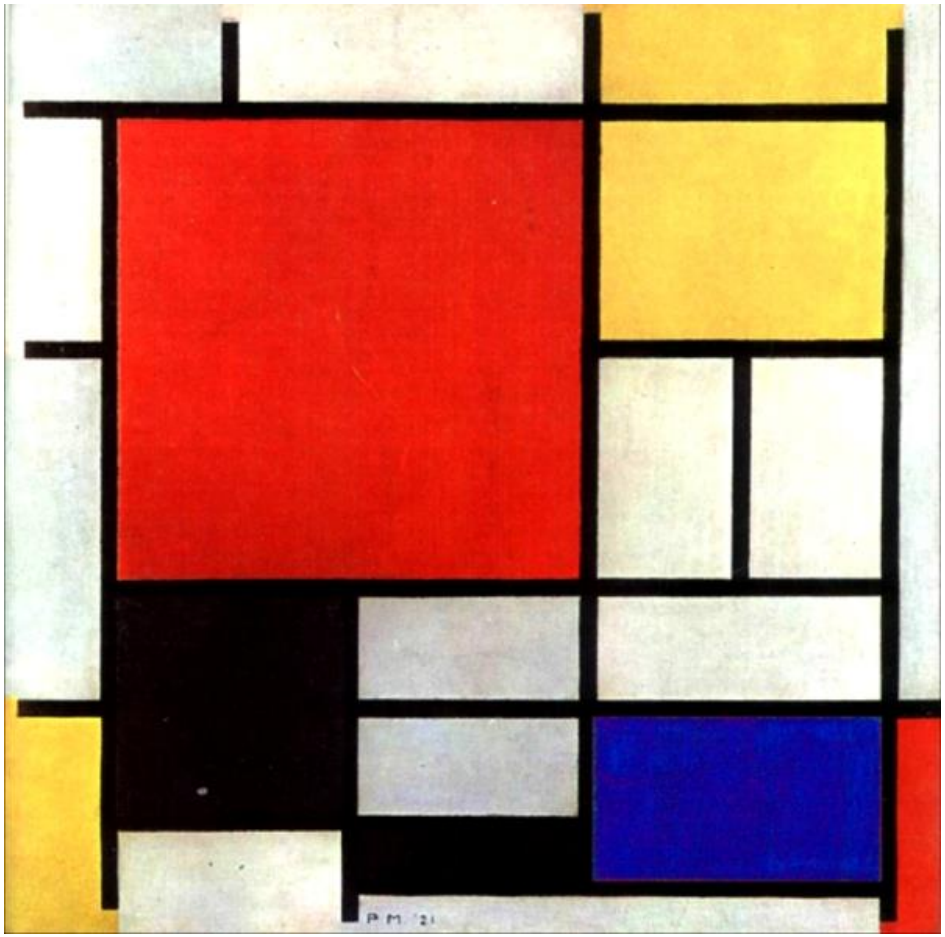

```
body { columns:3; column-gap:0.5in; }
img { float:page top right; width:3gr; }
```

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.

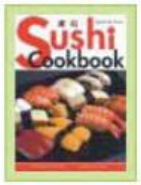
WHAT IS SUSHI?

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

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

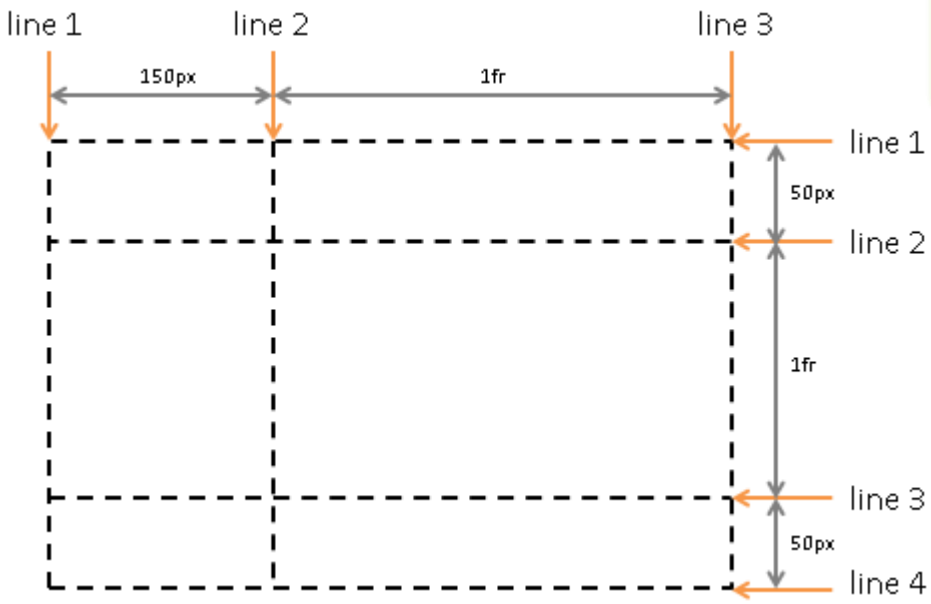
- Classic three-column layout

```
<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

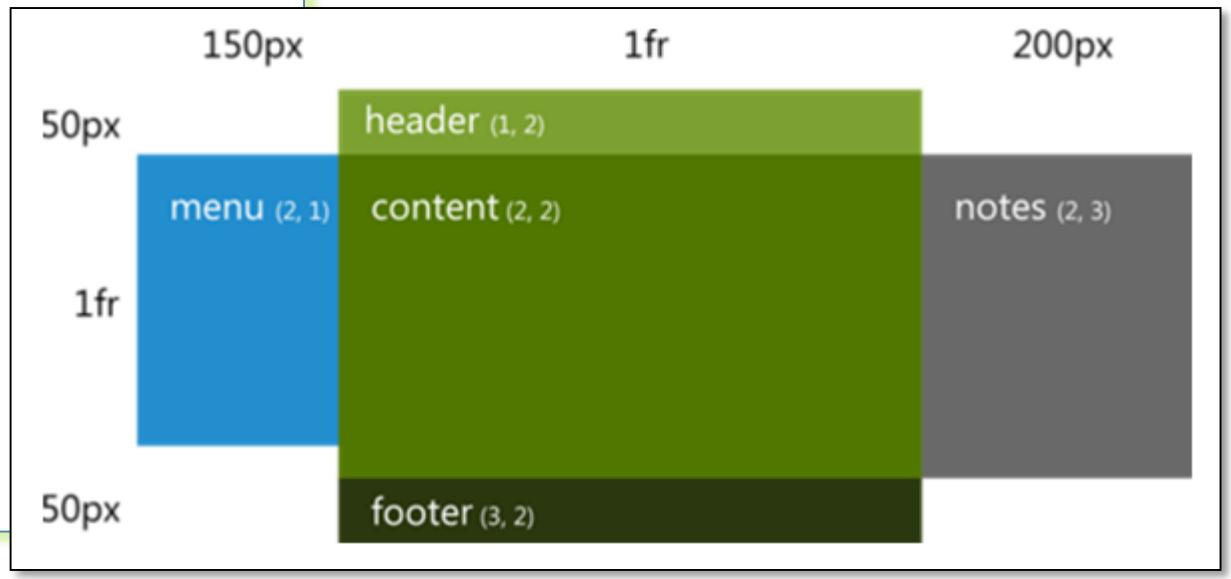
- Second step: to set how exactly the element will be placed on a grid it is necessary to specify to which line (both vertical and horizontal) it will be attached


```
#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

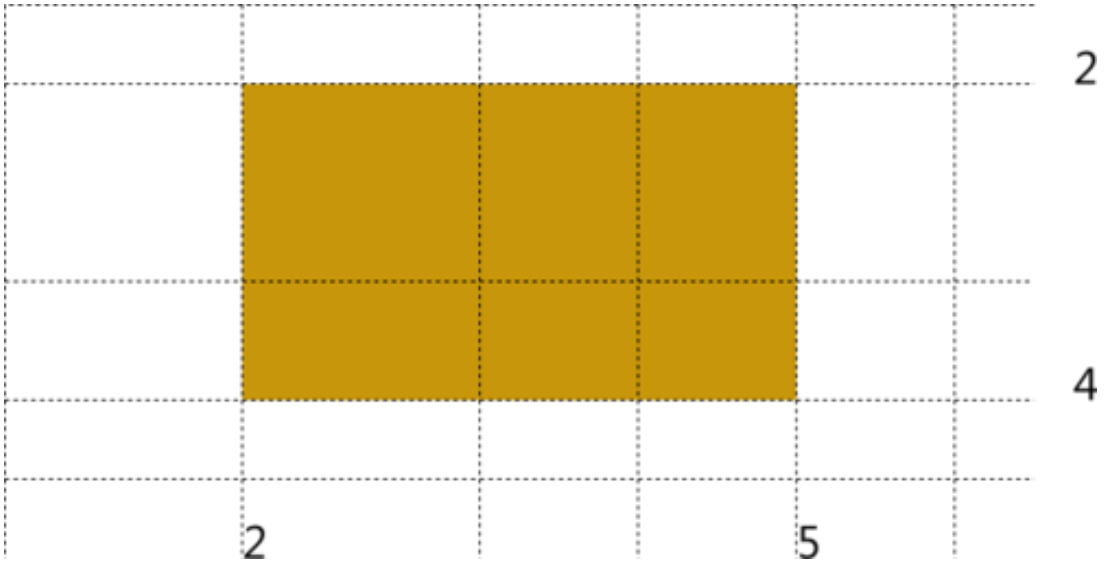


Fraction values

- 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; }
```

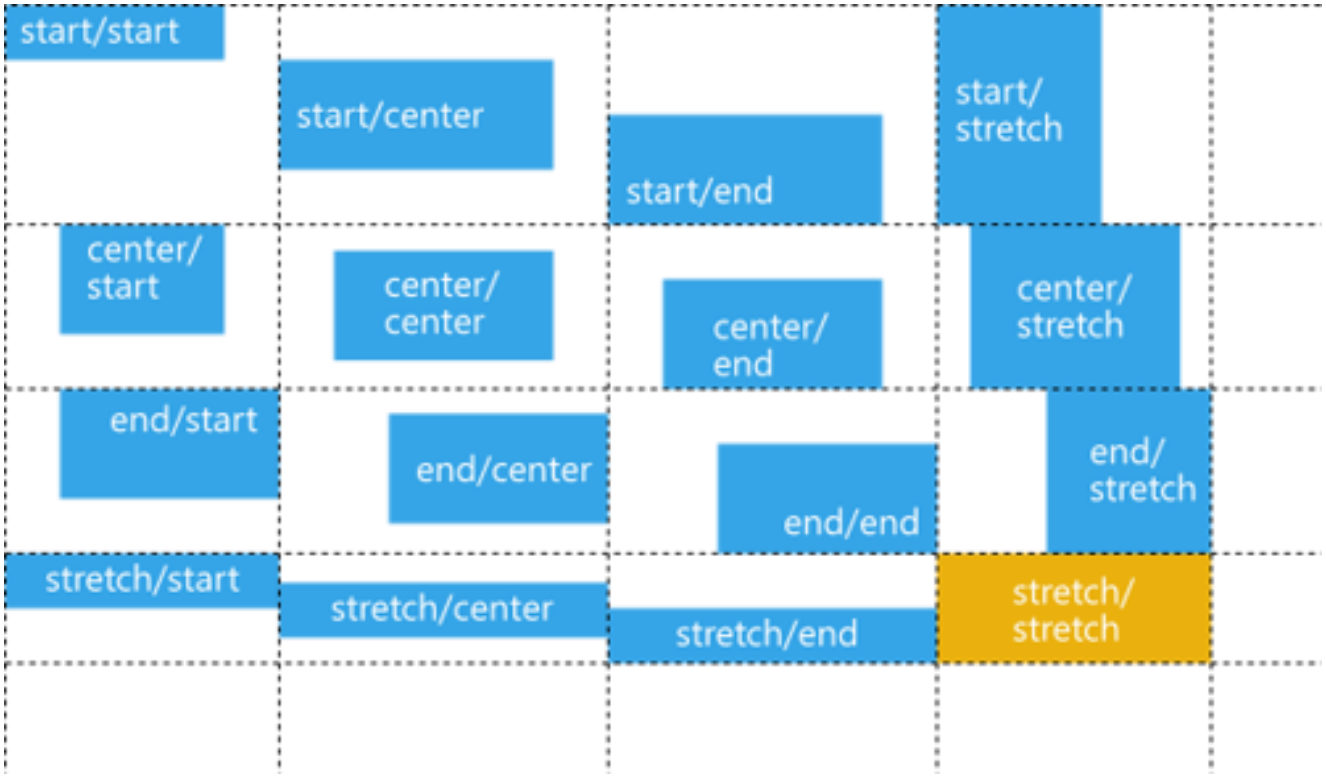
Repeated tracks

- Compact way to describe a repetitive grid layout

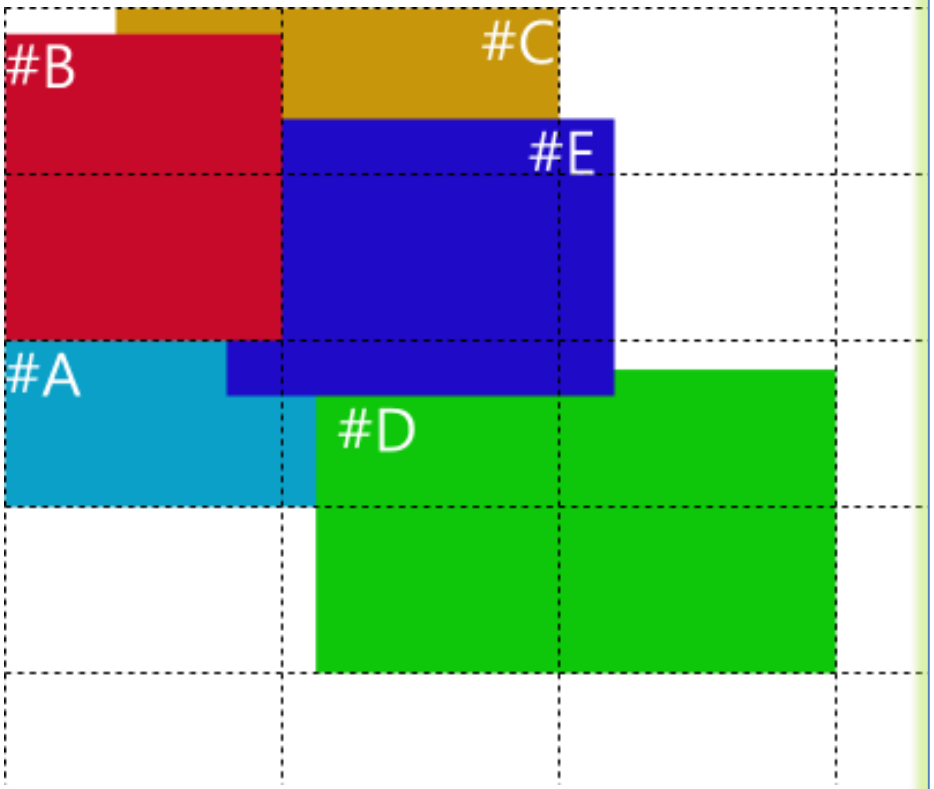
```
#grid {
  display: grid;
  grid-columns: 24px (120px 24px)[4];
  grid-rows: (1fr 24px)[3]; }
```

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; }
```

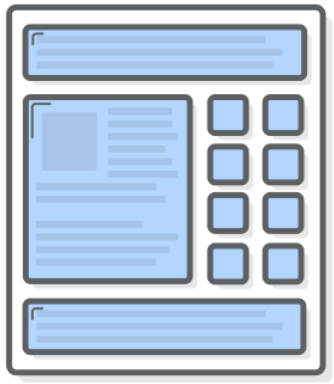
CSS FLEXBOX

Flexbox

- Alternative to floats for defining the overall appearance of a web page
  - Floats allow to horizontally position boxes
  - Flexbox gives complete control over the alignment, direction, order, and size of boxes
- Floats were originally intended for the magazine-style layouts

FLOATS

(MAGAZINE-STYLE LAYOUTS)



FLEXBOX

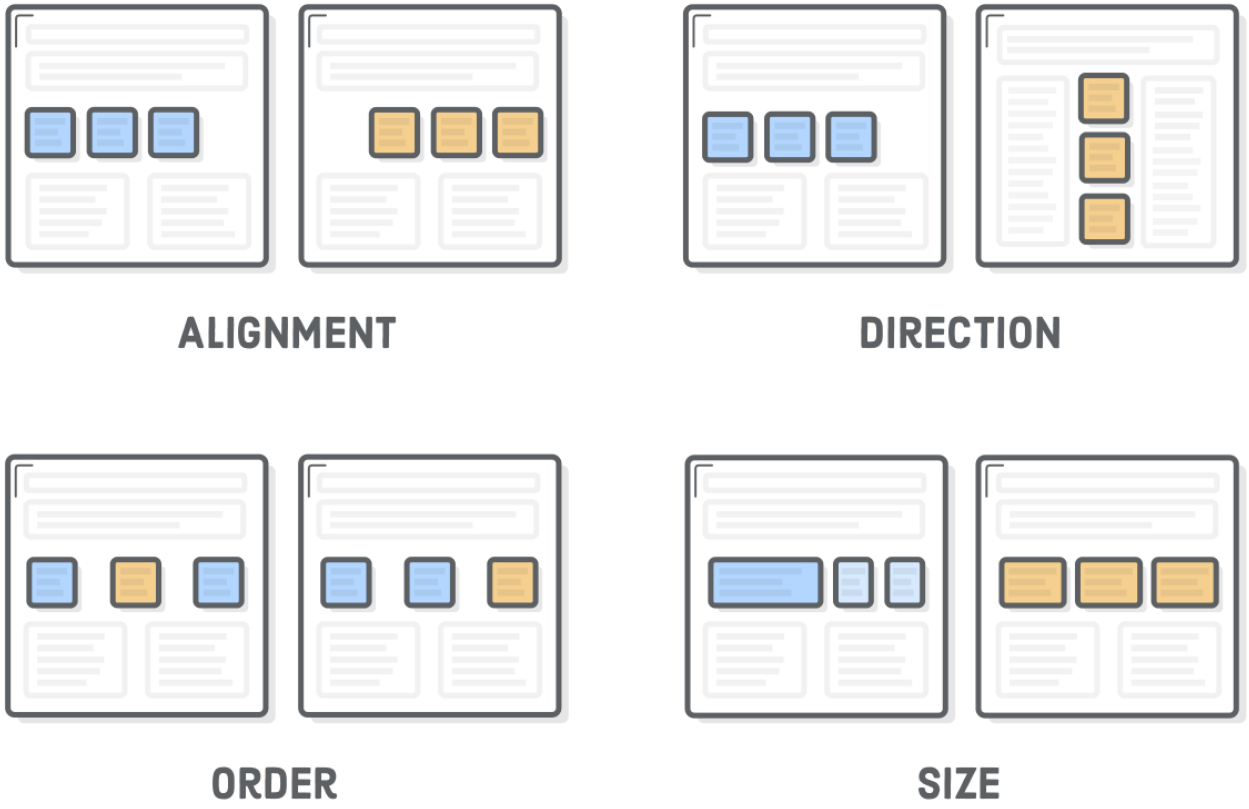
(OVERALL PAGE STRUCTURE)

Flexbox vs floats

- For the last decade or so floats were the sole option for laying out a complex web page
  - As a result, they're well supported even in legacy browsers, and developers have used them to build millions of web pages
- The kinds of layouts you can create with floats are actually somewhat limited
- Flexbox was invented to break out of these limitations
- We're finally at a point where browser support has hit critical mass and developers can start building full websites with flexbox

Flexbox

- https://internetingishard.com/html-and-css/flexbox/
- https://www.w3schools.com/css/css3_flexbox.asp



Flexbox

- Flexbox uses two types of boxes
  - Flex containers: their task is to group a set of flex items and define how they're positioned
  - Flex items
- Every HTML element that's a direct child of a flex container is an item

Horizontal alignment

- To turn one HTML elements into a flex container: display property set to “flex”
- “justify-content” property to define the horizontal alignment of its items
  - Values: center, flex-start, flex-end, space-around, space-between


```
.menu-container {
  /* ... */
  display: flex;
  justify-content: center;
}
```

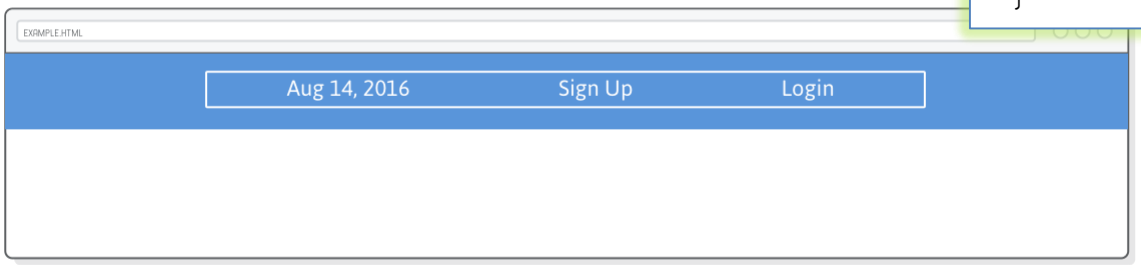
Distribution

- The justify-content property also lets you distribute items equally inside a container

SPACE-AROUND

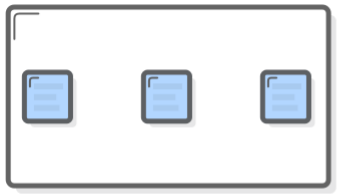
SPACE-BETWEEN

```
.menu {
  border: 1px solid #fff;
  width: 900px; display: flex;
  justify-content: space-around;
}
```



Grouping

- Flex containers only know how to position elements that are one level deep (i.e., their child elements)
  - You can group flex items using

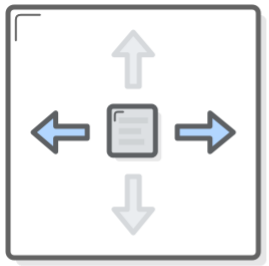


NO GROUPING
(3 FLEX ITEMS)

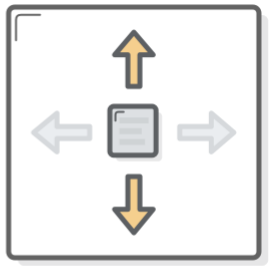
GROUPED ITEMS
(2 FLEX ITEMS)

Vertical alignment

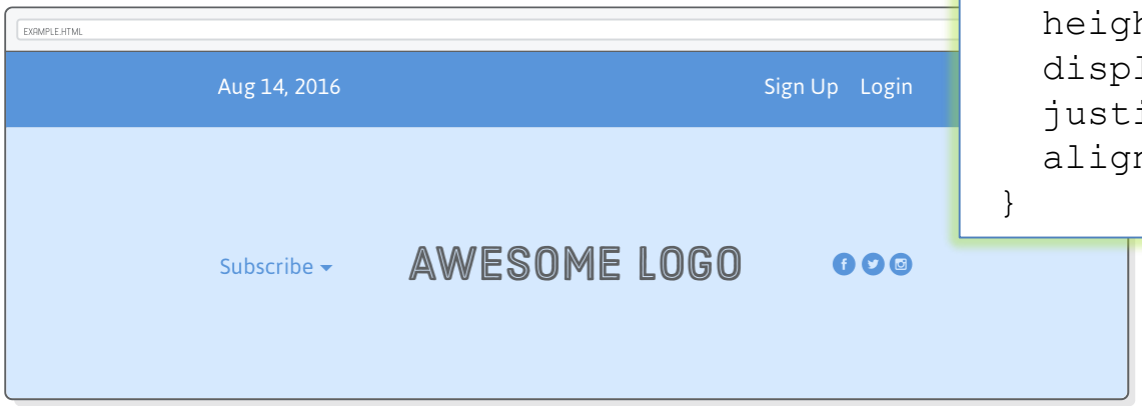
- Flex containers can also define the vertical alignment of their items



JUSTIFY-CONTENT



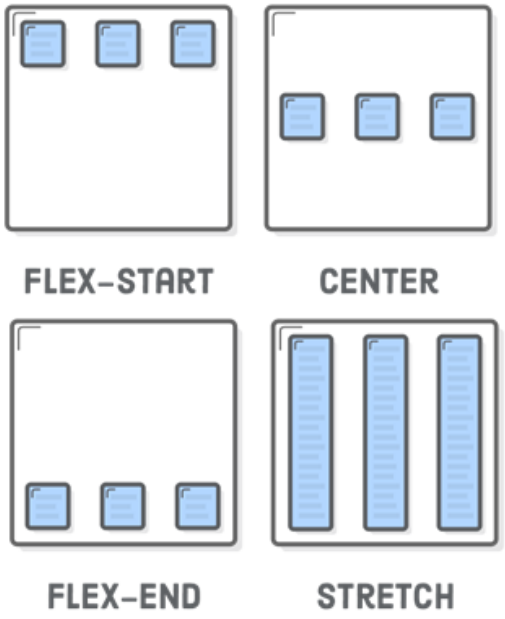
ALIGN-ITEMS

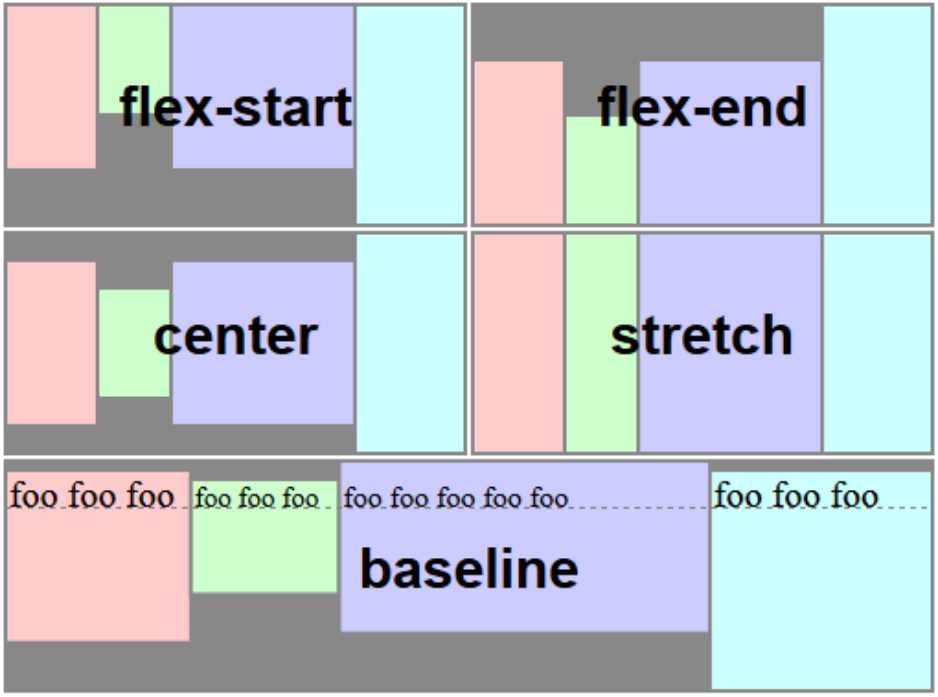



```
.header {
  width: 900px;
  height: 300px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Vertical alignment

- Values of align-items property: center, flex-start (top), flex-end (bottom), stretch, baseline



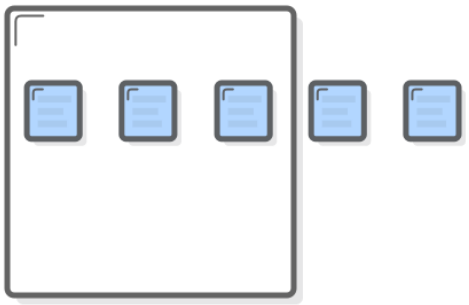


https://www.w3schools.com/cssref/css3_pr_align-items.asp

Wrapping

- The flex-wrap property creates a grid
  - Then, you can change alignment, direction, order, and size of items

```
.photo-grid {
  width: 900px;
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
}
```



NO WRAPPING

FLEX-WRAP: NOWRAP;

WITH WRAPPING

FLEX-WRAP: WRAP;

Direction

ROW

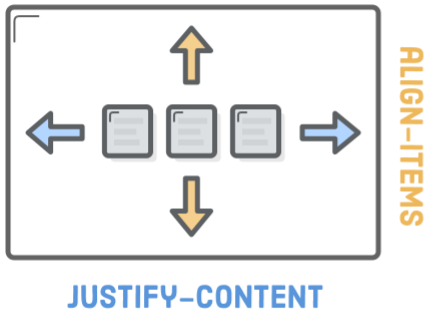
FLEX-DIRECTION: ROW;

COLUMN

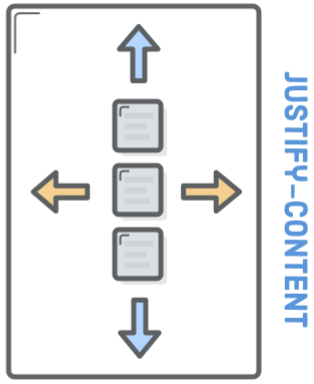
FLEX-DIRECTION: COLUMN;

- Refers to whether a container renders its items horizontally or vertically
  - Note: when you rotate the direction of a container, you also rotate the direction of the justify-content property: it refers to the container's vertical alignment, not horizontal alignment

FLEX-DIRECTION: ROW;



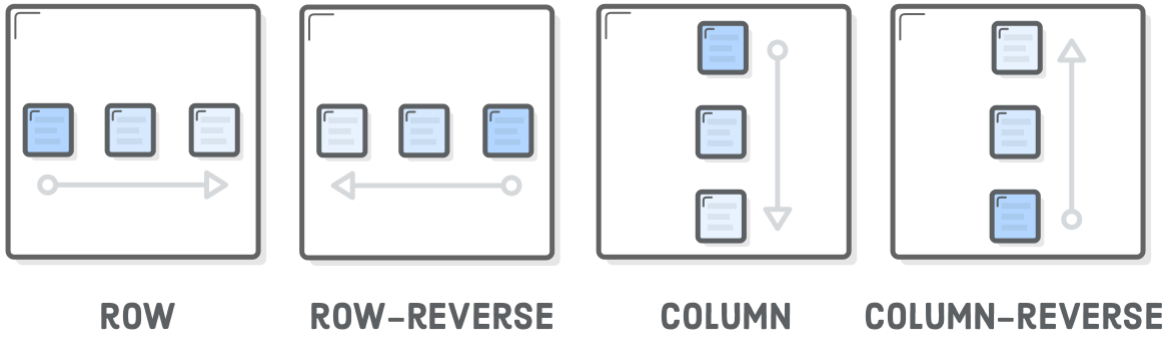
FLEX-DIRECTION: COLUMN;



Order

- The flex-direction property can also control the order in which items appear
  - row-reverse and column-reverse properties
- Can modify default HTML rendering

```
.photo-grid {
  width: 900px;
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
  flex-direction: row-reverse;
  align-items: center;
}
```

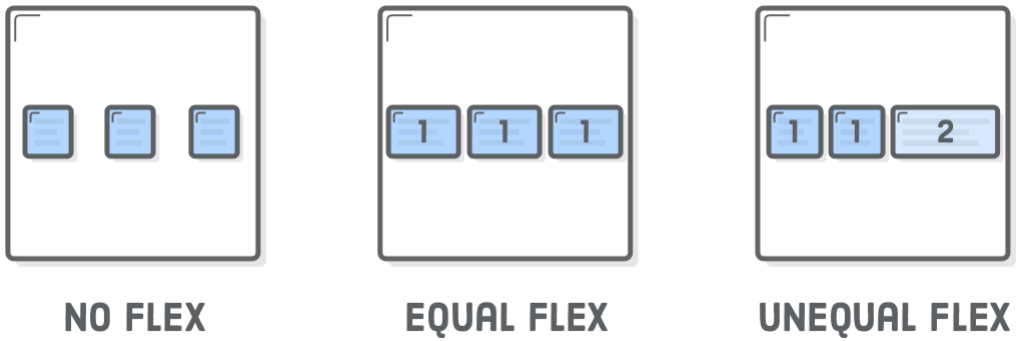


Flex items order and alignment

- It’s also possible to manipulate individual items
- order property
  - Default is 0, increasing or decreasing it moves the item to the right or left, respectively
- align-self property
  - Overrides the align-items value from its container
- Values: center, flex-start (top), flex-end (bottom), stretch, baseline

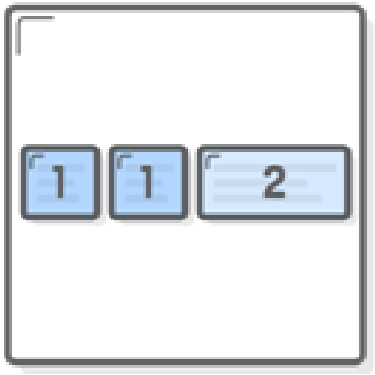
Flexible items

- Flex items are flexible: they can shrink and stretch to match the width of their containers
- The flex property defines the width of individual items in a flex container
  - It works as a weight that tells the flex container how to distribute extra space to each item
  - E.g., an item with a flex value of 2 will grow twice as fast as items with the default value of 1



Flexible items

- Example



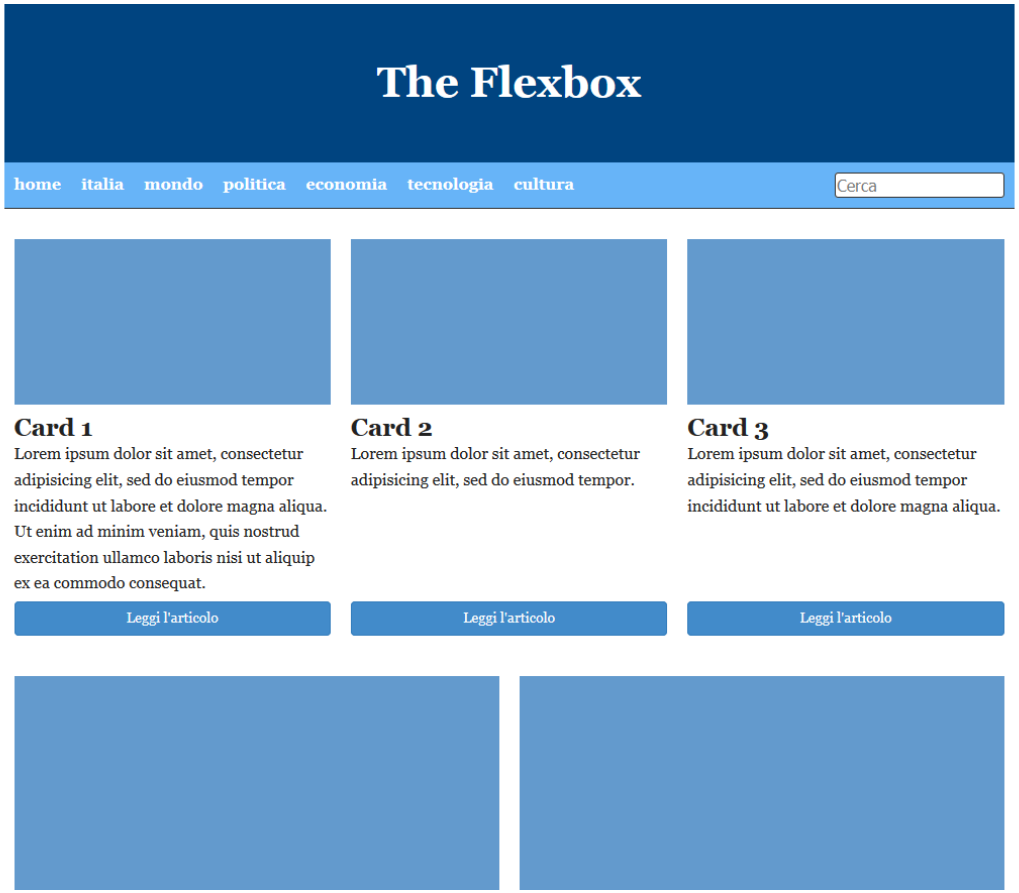

```
.footer {
  display: flex;
  justify-content: space-between;
}

.footer-item {
  border: 1px solid #fff;
  background-color: #D6E9FE;
  height: 200px;
  flex: 1; }

.footer-three { flex: 2; }
```

```
<div class='footer'>
  <div class='footer-item footer-one'></div>
  <div class='footer-item footer-two'></div>
  <div class='footer-item footer-three'></div>
</div>
```

Summary of CSS flexbox

- **`display: flex`** to create a flex container
- **`justify-content`** to define the horizontal alignment of items
- **`align-items`** to define the vertical alignment of items
- **`flex-direction`** if you need columns instead of rows
- **`row-reverse`** or **`column-reverse`** values to flip item order
- **`order`** to customize the order of individual elements
- **`align-self`** to vertically align individual items
- **`flex`** to create flexible boxes that can stretch and shrink

Grid layout with flexbox



- Example
  - http://www.html.it/guide/esempi/flexbox/8.%20colonne-wrap-flex.html

References for CSS flexbox

- Interneting is hard flexbox tutorial
  - https://internetingishard.com/html-and-css/flexbox/
- A complete guide to flexbox
  - https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- W3schools
  - https://www.w3schools.com/css/css3_flexbox.asp
- Flexbox, guida pratica
  - http://www.html.it/guide/flexbox-guida-pratica/

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/