CSS: Cascade Style Sheet

Chapter 3

A. CSS selectors and the cascade

- CSS3 selectors
- CSS cascade
  - Applying CSS to HTML
  - Specificity
  - Inheritance

CSS style rule


```
h1 { color: red; font-size: 2em; }
```

A **selector** specifies which elements the rule applies to.

Inside a pair of braces is a list of property declarations. A **property** refers to certain appearance of the element, and the **value** is the setting.

Selectors

- A simple selector consists of either a type selector or the universal selector followed by zero or more attribute selectors, id selectors, or class selectors.
  - Also include pseudo-classes and pseudo-elements
- A combined selector consists of two or more simple selectors separated by a combinator.
  - ' ', '>', '+', '~'

```
div#chap1 p.revised img {
   border: 1px solid red;
}
```

Type and universal selectors


```
em { color: red; }
```

- An type selector matches elements of a given type
  - E.g. set font color of em elements to red

```
* { padding: 0; margin: 0; }
```

- A universal selector matches any elements
  - E.g. reset padding and margins of all elements

id selectors


```
#first { color: blue; }
```

- An id selector matches a single element in an HTML doc by its 'id' attribute
  - **#first** is equivalent to ***#first**
- E#id matches a single element of type E that has the given ID
  - p#first selects the paragraph with the ID 'first'

class selectors


```
.info { color: red; }
```

- A class selector matches elements which belong to the class
  - **.info** is equivalent to ***.info**
  - A selector with both element type and class value, e.g. **p.info**, matches elements of that type that also belong to the class.
  - A selector can also specify more than one class value, e.g. **p.info.important**

Attribute selectors

- An attribute selector matches elements based on the value of an attribute
  - **E[attr]** matches elements E with the attribute
  - **E[attr="val"]** matches elements E whose given attribute *equals* to the value
  - **E[attr^="val"]** matches elements E whose given attribute *starts* with the value
  - **E[attr$="val"]** matches elements E whose given attribute *ends* with the value
  - **E[attr~="val"]** matches elements E whose given attribute *contains* the value

Examples

- Links to external web sites (absolute URL)

```
a[href^="http://"] { … }
<a href="http://www.gov.mo/">Macao gov</a>
```

- PNG images

```
img[src$=".png"] { … }
<img src="pic.png" alt="" />
```

- Input box of type 'password'

```
input[type="password"] { … }
<input type="password" name="pw" />
```

Pseudo-classes

- Pseudo-class enables selection of elements in a certain state or position in the DOM tree
  - Dynamic - :link, :visited, :focus, :active, :hover
  - Structural - :first-child, :nth-child( ), :last-child, etc
  - Negation - :not( )
  - :target
  - UI element states - :enabled, :disabled, :checked
- Ref: http://www.w3.org/TR/css3-selectors/
- Demo: http://www.quirksmode.org/css/contents.html

Dynamic pseudo-classes, 1


```
a:visited { color: #555555 }
```

- Select links that are visited or not
  - **a:link** matches an unvisited <a>
  - **a:visited** matches a visited <a>
- Only work for links
- A link can either be visited or unvisited, but not in both states at the same time

```
a:link { color: blue; }
a:visited { color: gray; }
```

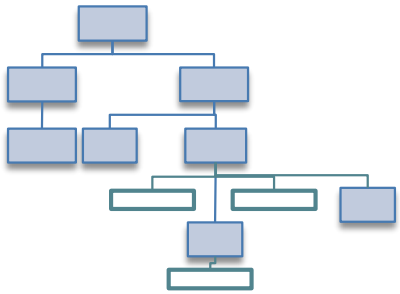
Dynamic pseudo-classes, 2


```
a:hover { text-decoration: underline; }
```

- Select elements based on user action
  - **a:hover** matches an <a> when the cursor is held over it
  - **a:active** matches an <a> when you click the link and do not release the mouse button
  - **a:focus** matches an <a> when it receives user focus
- :active and :focus also work on form controls
- :hover works on many elements like table rows, image

Structural pseudo-classes

- Match elements based on their position in the DOM tree
  - Only consider element children when determining the position



Position in the list of child elements	:first-child :last-child	:nth-child( ) :nth-last-child( )	:only-child
Position in the list of child elements of a certain type only	:first-of-type :last-of-type	:nth-of-type( ) :nth-last-of-type( )	:only-of-type
Others	:empty	:root	

Structural pseudo-classes


```
p:last-child { margin-bottom: 2em; }
```

- Select elements based on their position in the DOM tree
  - **E:first-child** matches any element E that is the first child of its parent
  - **E:last-child** matches any element E that is the last child of its parent
  - **E:nth-child(n)** matches any element E that is the nth child of its parent
  - **E:nth-last-child(n)** matches any element E that is the nth child of its parent, counting from the last child

Exercise

- Which elements do the following selectors match?

```
li:first-child
li:last-child
li:nth-child(2)
li:nth-last-child(3)
ol li:nth-child(1)
ul li:nth-last-child(1)
```

```
<body>
  <ol>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
    <li>five</li>
  </ol>
  <ul>
     <li>A</li>
     <li>B</li>
  </ul>
</body>
```

:nth-child(an+b)

- :nth-child() and :nth-last-child() also accept a formula of the form an+b
  - Selects the elements in the position x, where x = an+b, n=0,1,2,3,4, … (Ignore x <= 0)
- Examples:
  - :nth-child(2n+1) matches elements at position 1, 3, 5, 7, …. I.e. elements at odd position. You can also write :nth-child(odd)
  - :nth-child(2n) matches elements at position 2, 4, 6, 8, …. I.e. elements at even position. You can also write :nth-child(even)
  - :nth-child(3n) matches elements at position 3, 6, 9. I.e. every third element

More examples of :nth-child()

- :nth-child(n+5) matches elements at position 5,6,7,… i.e. elements not in the first 4 position
- :nth-child(-n+5) matches elements at position 1,2,3,4 and 5. I.e. the first 5 children
- :nth-last-child(-n+5) matches the last 5 children
- :nth-last-child(n+5) matches elements except the last 4children
- :nth-chlid(n+2):nth-last-child(n+2) matches elements except the first and last child
  - May also be written as :not(:first-child):not(:last-child)

:nth-of-type() ,.etc

- Four similar pseudo-classes count elements of the same type only
  - **E:first-of-type** matches any element E that is the first sibling of its type
  - **E:last-of-type** matches any element E that is the last sibling of its type
  - **E:nth-of-type(n)** matches any element E that is the nth sibling of its type
  - **E:nth-last-of-type(n)** matches any element E that is the nth sibling of its type, counting from the last child

Exercise

- Which elements do the following selectors match?
- Can you select each h2 element using :nth-child( ) ?

```
h2
h2:first-of-type
h2:last-of-type
h2:nth-of-type(2)
p:nth-of-type(even)
```

```
<body>
  <h1>..</h1>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
</body>
```

Example

- Add divider between consecutive

```
div:nth-child(n+2) {
  border-top: 1px dashed green;
}
```

OR

```
div:nth-last-child(n+2) {
  border-bottom: 1px dashed green;
}
```

```
<body>
  <div>first</div>
  <div>second</div>
  <div>third</div>
  <div>fourth</div>
</body>
```

First div

Second div

Third div

Fourth div

Negation E:not(s)

- Matches any E element that does not match the simple selector s.
  - p:not(:first-child) matches p that is not a first child
  - p:not(.important) matches p that is not in the class 'important'
  - :not(p) matches any elements other than p (e.g. div, em)
  - img:not([src$="png"]) matches img of format other than png (e.g. gif, jpg)
  - p:not(:first-child):not(:last-child) matches all p except the first and last child

:target

- An element with an id matches the :target pseudo-class if the URL of the current web page has an anchor referring to that id

```
<style>div:target { color: red; } </style>

<body>
  <div id='d1'>first</div>
  <div id='d2'>second</div>
  <div id='d3'>third</div>
</body>
```

Which will be in red if the URL of current page is http://example.com/page.htm#d2 ?

Pseudo-element selectors

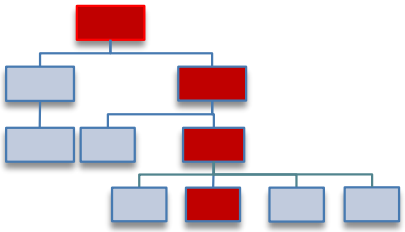
- Pseudo-element selectors match part of an element
  - p::first-line matches the first line of the <p>
  - p::first-letter matches the first letter of the <p>
  - CSS2 uses the syntax p:first-line and p:first-letter
  - Ref and examples:
    - http://www.w3.org/TR/css3-selectors/#pseudo-elements
    - http://www.w3schools.com/css/css_pseudo_elements.asp

Generated content

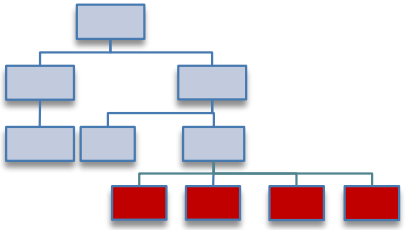
- Two special pseudo-elements ::before and ::after allows adding content before or after some elements
  - E.g. p.note::before { content: "Note: " } inserts "Note: " before the content of every paragraph in the class 'note'.
  - E.g. p:last-child::after { content: url(signature.png) } inserts an image at the end of the last paragraph
  - For more examples, search 'css generated content'

Combined selectors

- Combine two or more simple selectors
- Four kinds:
  - Descendant selectors: E1 E2
  - Child selector: E1 > E2



  - Adjacent sibling: E1 + E2
  - General sibling: E1 ~ E2



Descendant selectors


```
ul li { list-style-type: square; }
```

- Select elements that are descendants of another element
- Form: **E1 E2**, where E1 is a simple selector for the ancestor, and E2 is a simple selector for the descendant.
- Note: A space character separates the simple selectors in descendant selectors. There is no space within a simple selector. E.g. "p.revised" is different from "p .revised"

Example

- Note that there may be more than one elements between the ancestor and descendants.

```
div#toc span.attr { color: blue; }
…
<div id="toc">
  <h1>Core attributes</h1>
  <ul>
    <li><span class="attr">id</span></li>
    <li><span class="attr">class</span></li>
  </ul>
</div>
```

Child selectors


```
ul>li { border-left: 1px solid red; }
```

- Select elements that are direct child of another element
- Form: **E1 > E2**, where E1 is a simple selector for the parent, and E2 is a simple selector for the child.

Example

- Compare the difference between **ul>li** and **ul li**
- How to select the list items in the nested ol?

```
ul>li { border-left: 1px solid red; }
...
<!-- an ordered list inside an unordered list -->
<ul>
    <li>...</li>
    <li>...</li>
    <li>
      <ol><li>...</li> <li>...</li></ol>
    </li>
</ul>
```

Adjacent sibling selectors


```
h2+h3 { margin-top: -5mm; }
```

- Select a sibling element that is immediately preceded by another element
- Form: **E1 + E2**, where E1 is a simple selector for the preceding sibling, and E2 is a simple selector for the element to be selected. Both elements must have the same parent.

General sibling selectors


```
h1~p { font-size: larger; }
```

- Select a sibling element that is preceded by another element
- Form: **E1 ~ E2**, where E1 is a simple selector for the preceding sibling, and E2 is a simple selector for the element to be selected. Both elements must have the same parent. *There may be other siblings between E1 and E2.*

Example

- Compare the difference between **h1+h2** and **h1~h2** in this example

```
h1+h2 { margin-top: -5mm  }
…
<h1>Core attributes</h1>
<h2>The id attribute</h2>
<p> … </p>
<h2>The class attribute</h2>
<p> …</p>
```

Grouping selectors

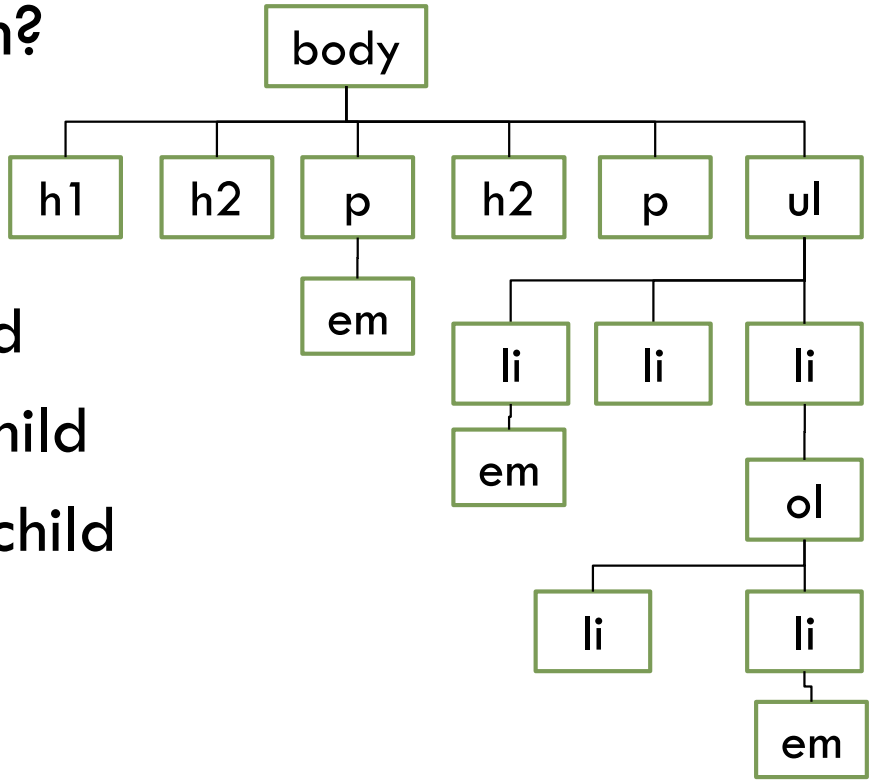
- You can group selectors using comma

```
#summary { color: red; }
.attr { color: red; }
h1 { color: red; }
```

```
#summary, .attr, h1 { color: red; }
```

Exercise

- Which elements do the following selectors match?
  - li em
  - ul li
  - ul>li
  - li li
  - h1+h2
  - h1~h2
  - ul * em
  - ul>*>em
  - li:first-child
  - ul li:first-child
  - ul>li:first-child
  - h2+p em
  - h1+h2+p
  - h2~ul ol em



Summary: selectors

Selectors	pattern
Type selector	E
id selector	#id
Class selector	.class
Universal selector	*
Attribute selector	E[attr], E[attr=val], …
Pseudo-class	:first-child, :nth-child( ), …
Pseudo-element	::first-letter, ::first-line, …
Descendant selector	E1 E2
Child selector	E1>E2
Adjacent sibling selector	E1+E2
General sibling selector	E1~E2

Conflicting style declarations

- Several style rules that match an element may specify conflicting values for a property

```
li { color: red; }
li li { color: green; }
li.hilte { color: blue; }

<ul>
    <li>one</li>
    <li>two:
        <ul><li>a</li><li>b</li>
            <li class="hilte">c</li></ul>
    </li>
    <li>three</li>
</ul>
```

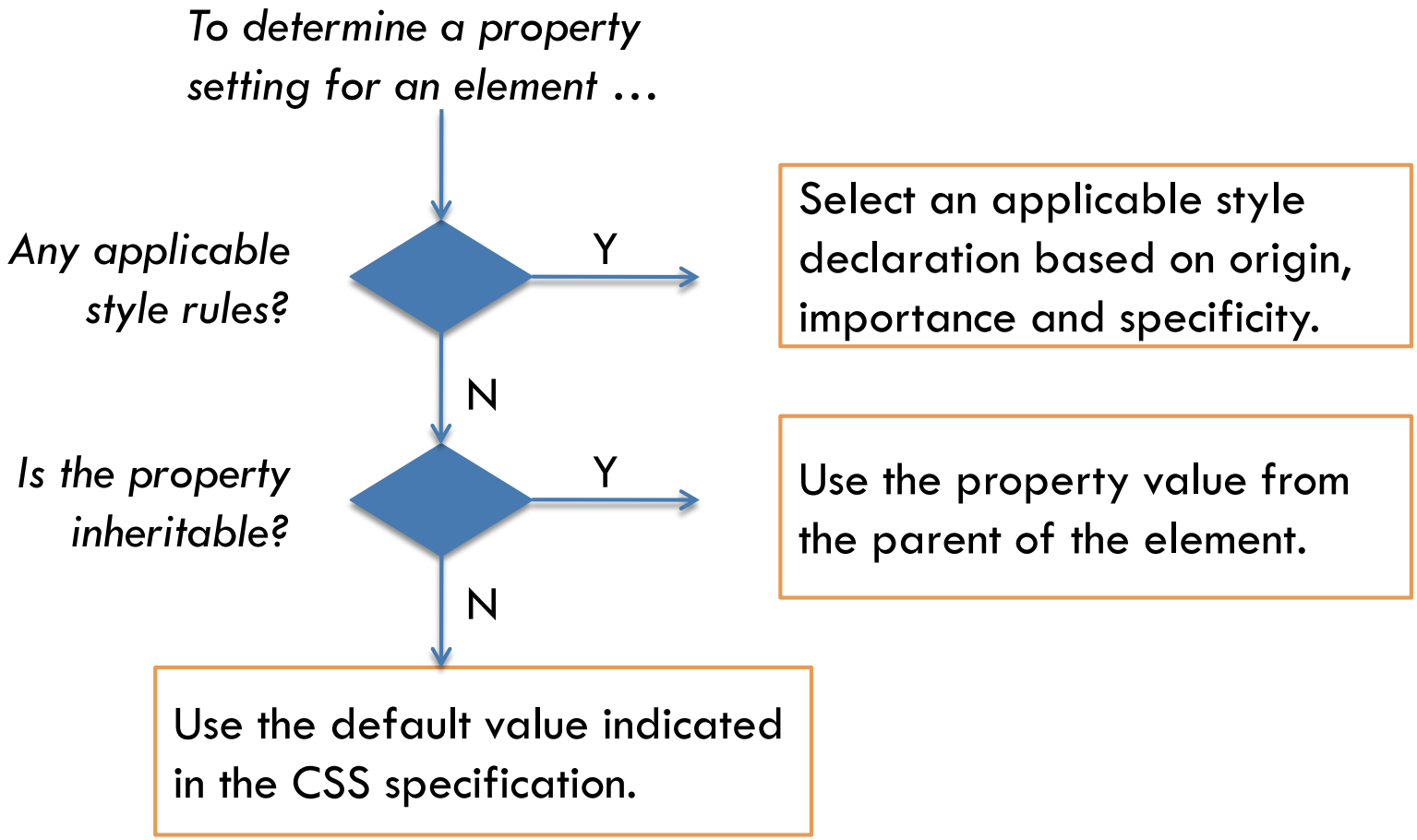
The CSS cascade

- The CSS cascade determines which style declaration wins in case of conflicting values for a property.
- Ref:
  - http://www.maxdesign.com.au/articles/css-cascade/
  - http://www.w3.org/TR/CSS2/cascade.html

The CSS Cascade

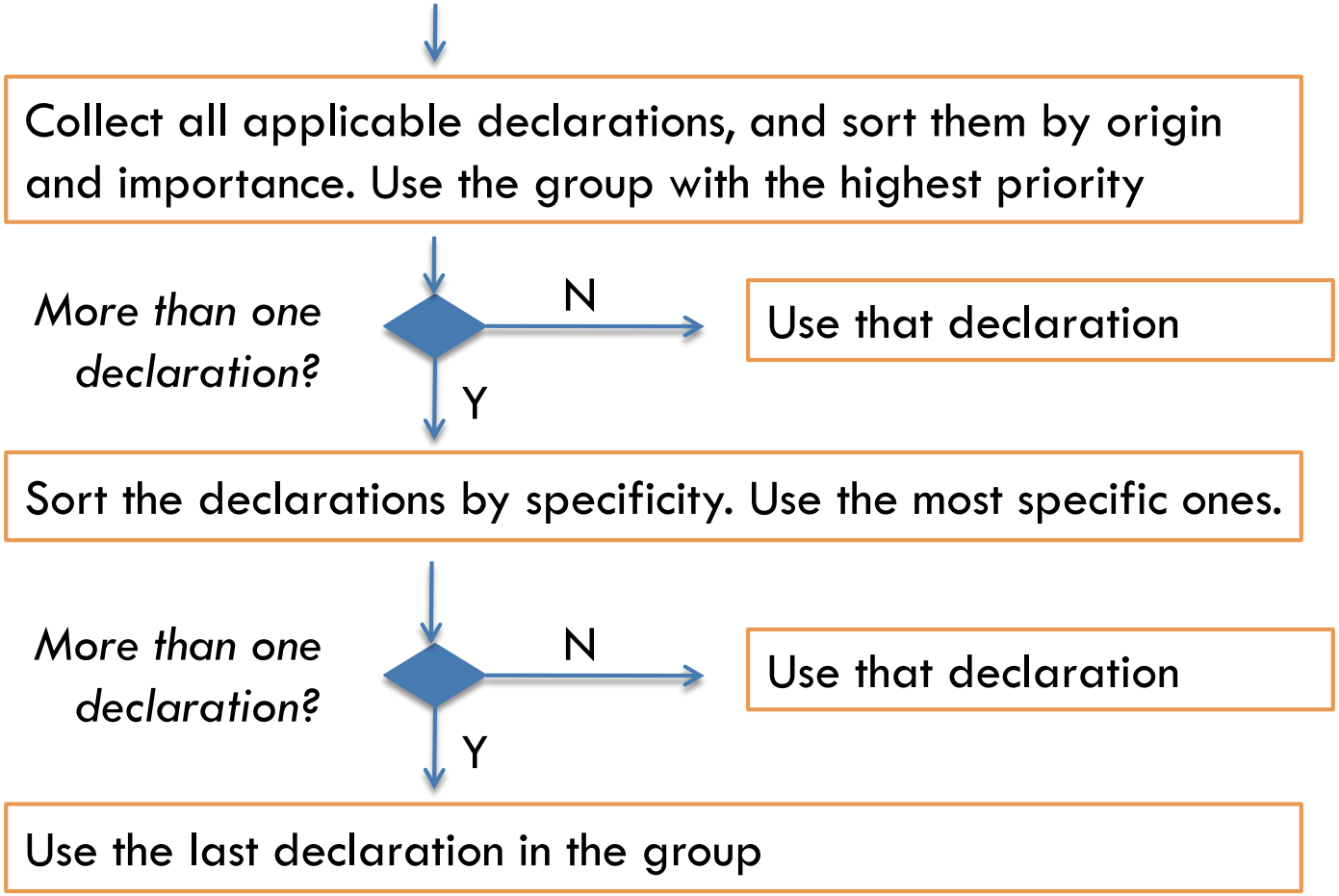
- To display a web page, the browser collects style declaration for an element from various origins.
- If there are more than one applicable declaration for a CSS property for an element, the cascade has to decide which declaration to apply.
- If there are no applicable declaration, the browser uses inherited value or default value as indicated in the spec.

CSS Cascade


To determine a property setting for an element …
Any applicable style rules?
Y
Select an applicable style declaration based on origin, importance and specificity.
N
Is the property inheritable?
Y
Use the property value from the parent of the element.
N
Use the default value indicated in the CSS specification.

Select an applicable declaration

To select an applicable style declaration…



Origins of style declaration

- A style declaration comes from 3 possible origins
  - Browser style sheet (user agent style sheet)
    - Each browser has a default style sheet
  - Author style sheet
    - Added by the author of the page
    - Inline, embedded, external, @import
  - User style sheet
    - E.g. Google Chrome allows a user to change the style of a web page
    - We'll ignore this in later discussion

Browser style sheet

- Each browser comes with a default style sheet
  - Ref. http://meiert.com/en/blog/20070922/user-agent-style-sheets/
- Defines the basic style to show an HTML without any style sheets.
- Different browsers have different default
  - E.g. default style for <a>: Firefox uses underline, whereas Chrome uses no underline
- To ensure consistent presentation, you may use a reset style sheet to override the browser style sheet with the same basic styles

Author style sheet

- The author of an HTML doc can specify style rules in three ways:
  - Inline style
  - Embedded style / internal style
  - External style

Inline style

- style attribute inside the start tag of an element
- Style applies to one element only
- Higher priority than embedded and external style declarations
- Mix up presentation and content.
  - May use it for quick testing. But not recommended in production version

```
<h1 style="color: red">HTML essential</h1>
…
<h1>CSS essential</h1>
```

Embedded style

- keep all rules in a <style> element inside <head>
- Style applies to one HTML doc only

```
<head>…
  <style> h1 { color: red; } </style>
</head>
<body>
  <h1>HTML essential</h1>
  <p>some text</p>
  <h1>CSS essential</h1>
</body>
```

External style

- keep all rules in a separate file and reference it in a **<link>** element
- Several HTML files can link to the same style sheet. Consistent style
- An external style sheet can also import another external style sheet with @import

```
<head>…
  <link rel="stylesheet" href="web.css"/>
</head>
<body>
  <h1>HTML essential</h1>
  <p>some text</p>
  <h1>CSS essential</h1>
</body>
```

```
/* web.css */
h1 { color: red; }
```

Importance

- An important declaration ends with !important
- An important declaration wins over a normal declaration

```
<head>…
  <style>
      #title { color: blue; }
      h1 { color: red !important; }
  </style>
</head>
<body>
  <h1 id='title'>HTML essential</h1>
  <p>some text</p>
</body>
```

Origin and importance

- Style declarations that match an element may come from several sources, in decreasing order of priority
  - Author style sheet with !important
  - Author style sheet
  - Browser style sheet
- A property declaration with higher priority wins.
- If two property declarations have the same priority, consider the specificity

Example

- What style will be used for the four links? Explain.

```
/* user agent style sheet */
a { text-decoration: underline }
```

```
<style>
   a.plain { text-decoration: none }
   a.broken {text-decoration: line-through !important }
</style>

 <a href='p1.htm'>link 1</a>
 <a href='p2.htm' class='plain'>link 2</a>
 <a href='p3.htm' class='broken'>link 3</a>
 <a href='p4.htm' class='plain broken'>link 4</a>
```

Specificity

- Specificity of a CSS selector is a list of three numbers a-b-c
  - a is the number of ID selectors
  - b is the total number of class selectors, attributes selectors, and pseudo-classes
  - c is the total number of type selectors and pseudo-elements
  - Ignore the universal selectors
  - Selectors in :not() is counted, but :not() itself is not counted as a pseudo-class

Example

Selectors	specificity
em	0-0-1
div p em	0-0-3
p.revise	0-1-1
p.revise:first-of-type	0-2-1
a.plain[href^='http://']	0-2-1
#main	1-0-0
p#main	1-0-1

Selectors	specificity
ul#toc > li	1-0-2
ul#toc > li:nth-child(odd)	1-1-2
p::first-line	0-0-2
div:not(.important)	0-1-1
img:not([alt])	0-1-1
div>*>em	0-0-2
#toc > :first-child	1-1-0

Comparing specificity

- When there are more than one applicable style declaration, use the ones with the highest specificity
  - Example: suppose selector 1 has specificity a1-b1-c1 and selector2 has a2-b2-c2
  - If a1>a2, selector 1 wins. If a2>a1, selector 2 wins.
  - If a1=a2, compare b1 and b2. If b1>b2, selector 1 wins. If b2>b1, selector b2 wins.
  - If a1=a2 and b1=b2, compare c1 and c2. The higher wins.

Example: specificity

Selectors	Specificity
div#content p#p2 span.note	high
div.redbox p#p2 span.note	
div p#p2 span.note	
div p:nth-child(2) span.note	
div p span.note	
div span.note	
div p span	
div span	
span	low

Exercise: compare the specificity of the following:

p:nth-child(2) span
a[href$='gif'] span
div#content p span
p > span.note
p > * > span.note

Example

- Determine the color of text in the nested list

```
li { color: red; }
li li { color: green; }
li.hilte { color: blue; }
```

```
li { color: yellow;  }
li li { color: black !important; }
```

```
<ul>
    <li>one</li>
    <li>two:
        <ul><li>a</li><li>b</li>
            <li class="hilte">c</li></ul>
    </li>
    <li>three</li>
</ul>
```

Inheritance and default value

- In some cases, no style declaration for a property applies to an element
- If the property is inherited, the element obtains the property setting from its parent
  - E.g. most typographical properties
- If the property is *not* inherited, the element uses the default value
  - Ref: http://www.w3schools.com/CSS/css_reference.asp

Property	Default value
background-color	transparent
width	auto (stretch to fill container)
position	static (normal flow in chap 5)

Example


```
body { color: black }
em { color: blue; }
div { width: 400px; }
div p { margin: 50px 50px; }

<body>
  <div>
     <p>first paragraph</p>
     <p><em>second</em> paragraph</p>
  </div>
</body>
```

Both <p> inherit the black text color, while does not.

There is no style declaration for width of the two <p>. However, they do not inherit the width (400px) from their parent. Instead, they take the default value 'auto'.

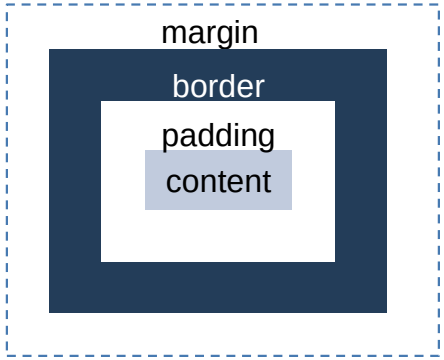
CSS Layout Outline

- A. Normal flow
- B. Floating
- C. Web page layout
- D. Absolute positioning
- E. Flexible box

CSS Box

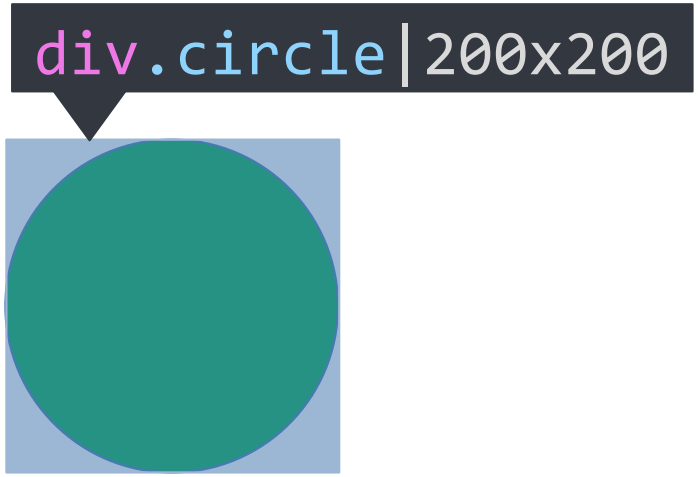
- Every element is displayed in one (or more) box
  - We say that this element *generates* the box
- The box has content, padding, border and margin.

```
<body>
  <h1>Structure of HTML</h1>
  <p>An HTML doc has a
     <em>tree structure</em>,
     where a <span>parent</span> element
     contains some <span>children</span>
     elements.
  </p>
</body>
```



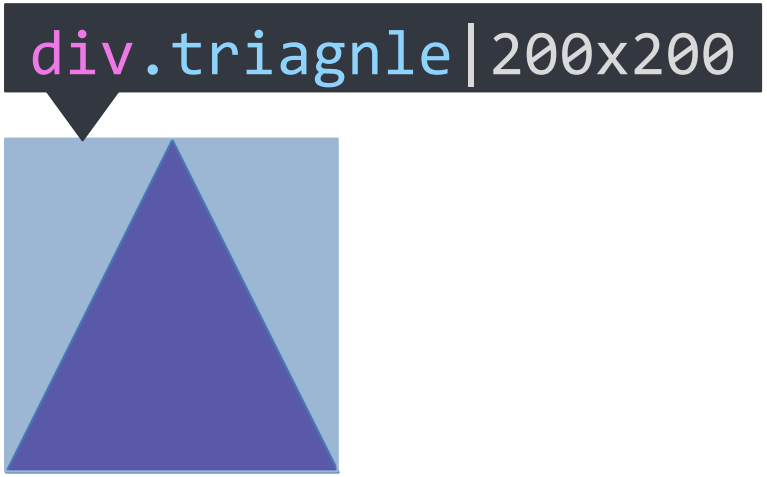
Boxes Everywhere!

- Everything in HTML is a box.



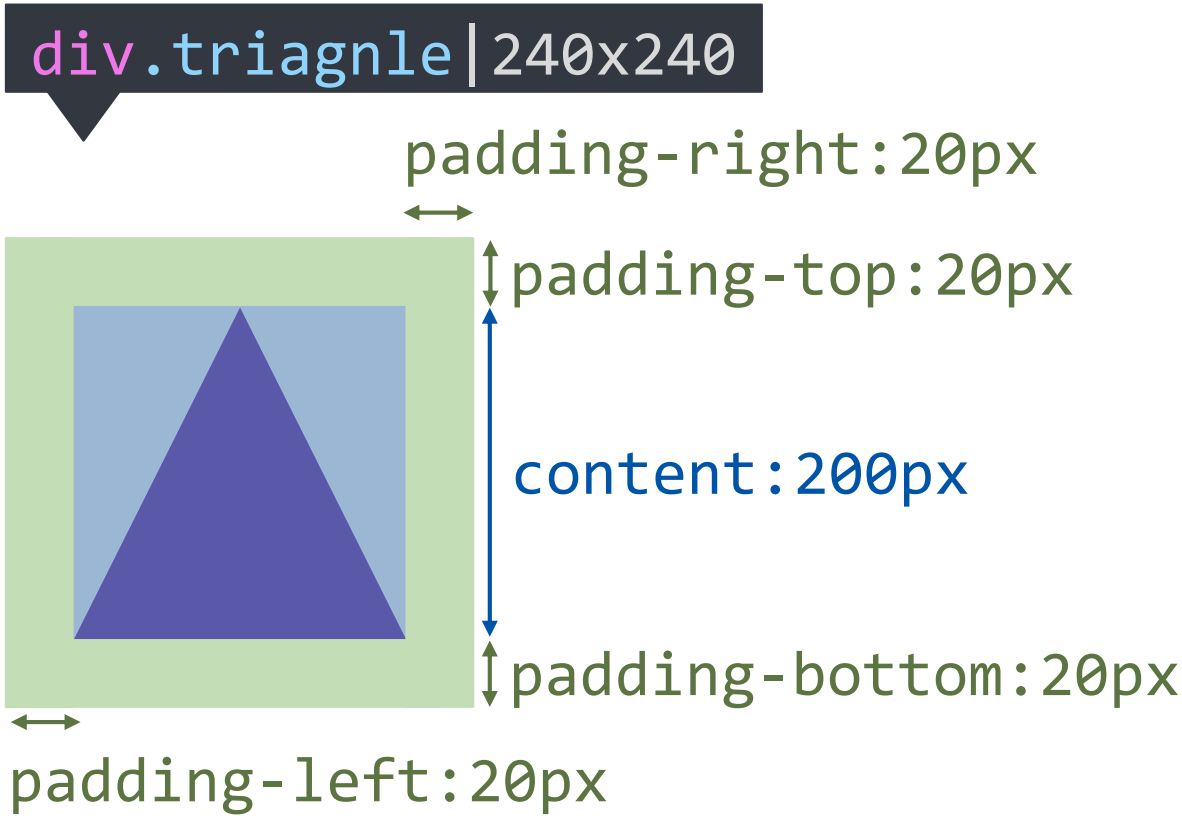
Boxes Everywhere!

- Everything in HTML is a box.



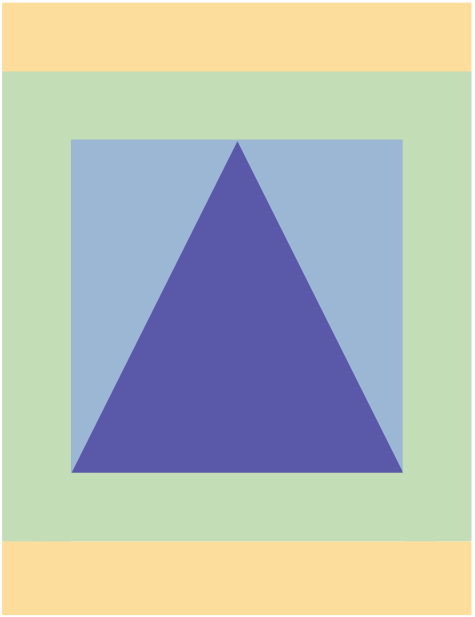
Box View

- content
- border
- padding
- margin



Box View

div.triagnle|240x280



border-top:20px

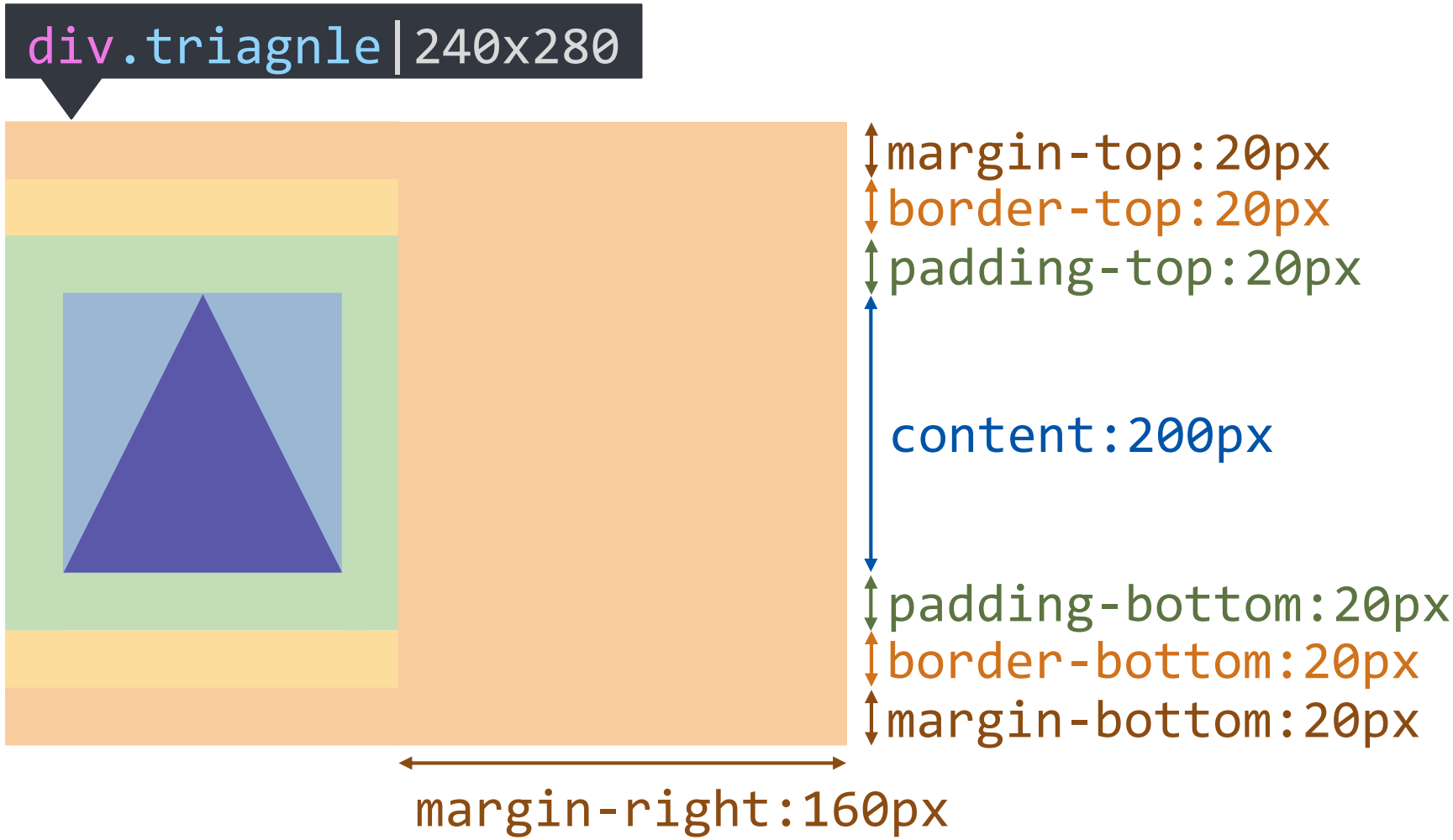
padding-top:20px

content:200px

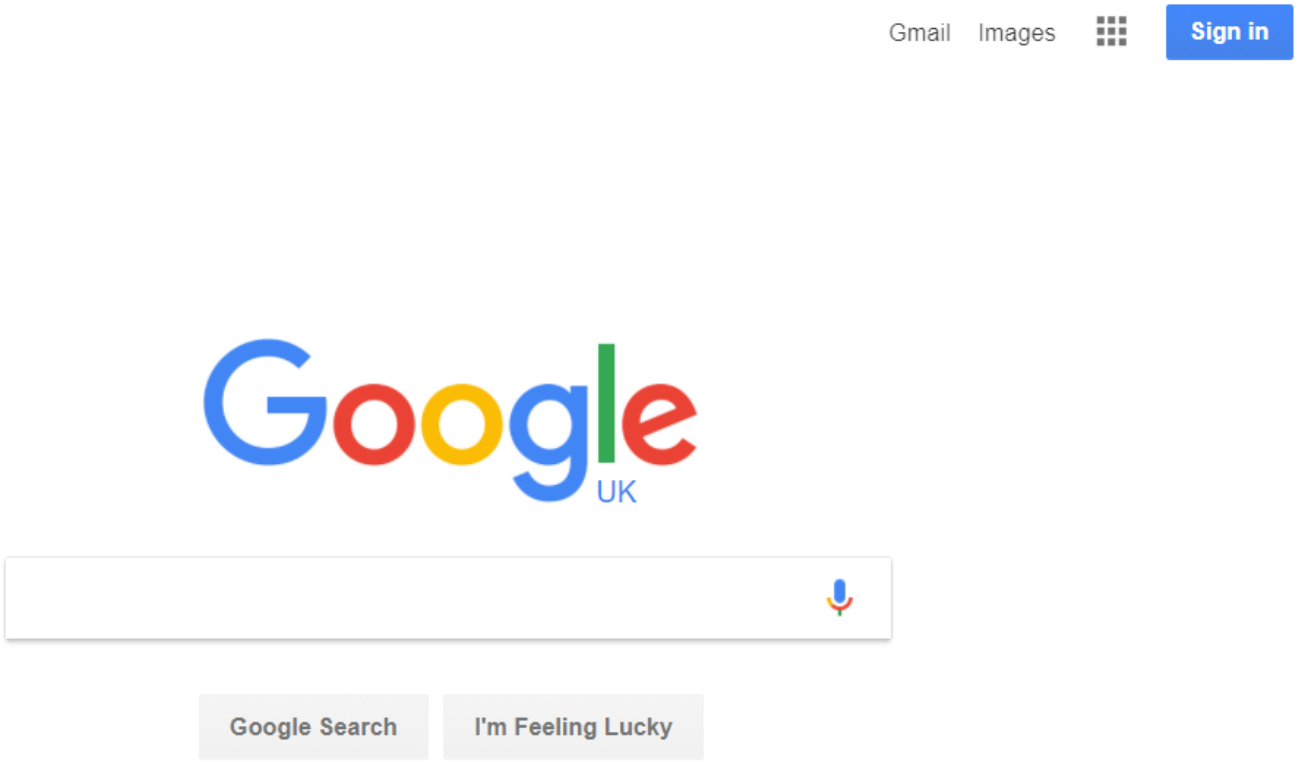
padding-bottom:20px

border-bottom:20px

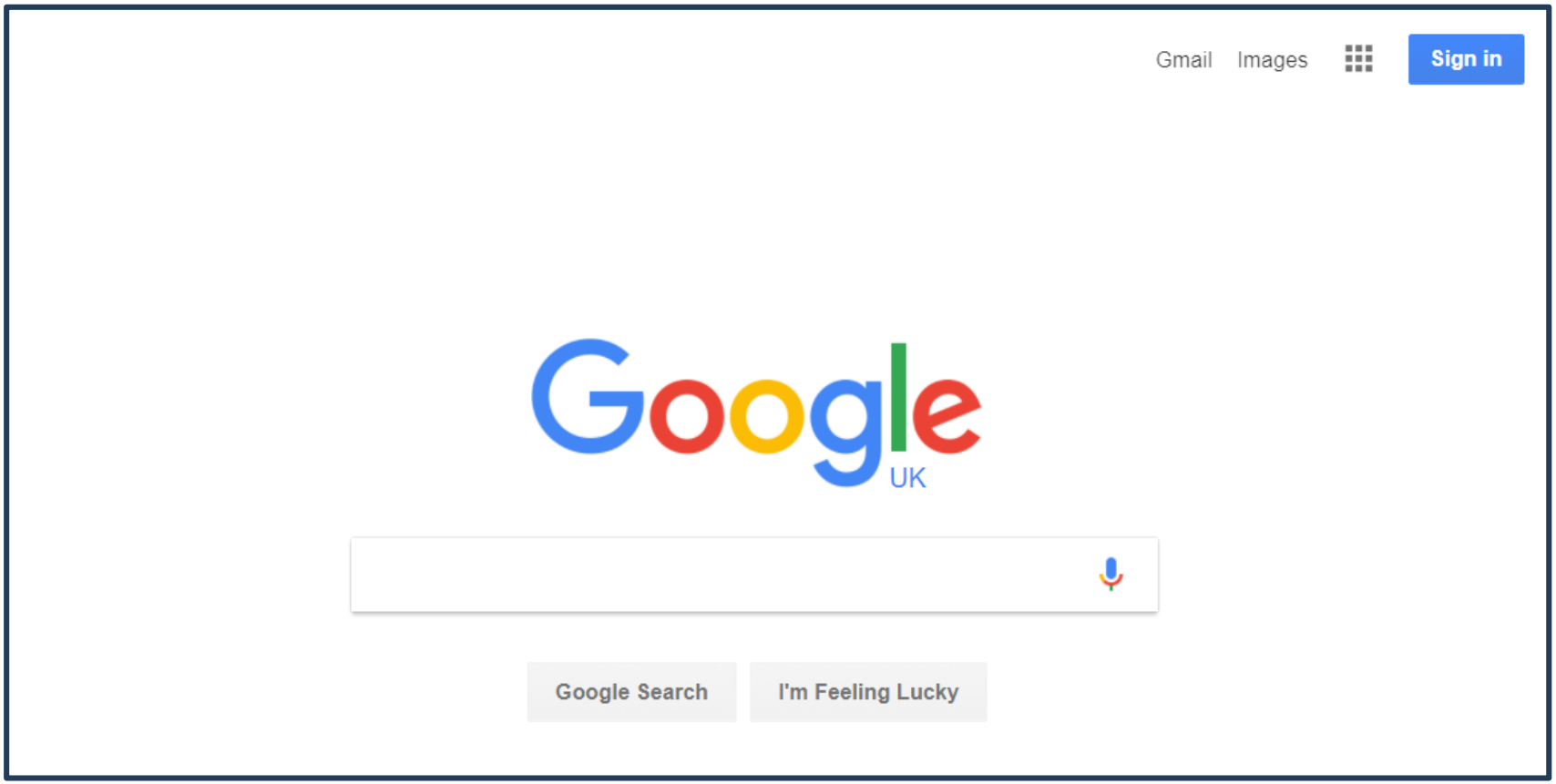
Box View


div.triagnle|240x280
margin-top:20px
border-top:20px
padding-top:20px
content:200px
padding-bottom:20px
border-bottom:20px
margin-bottom:20px
margin-right:160px

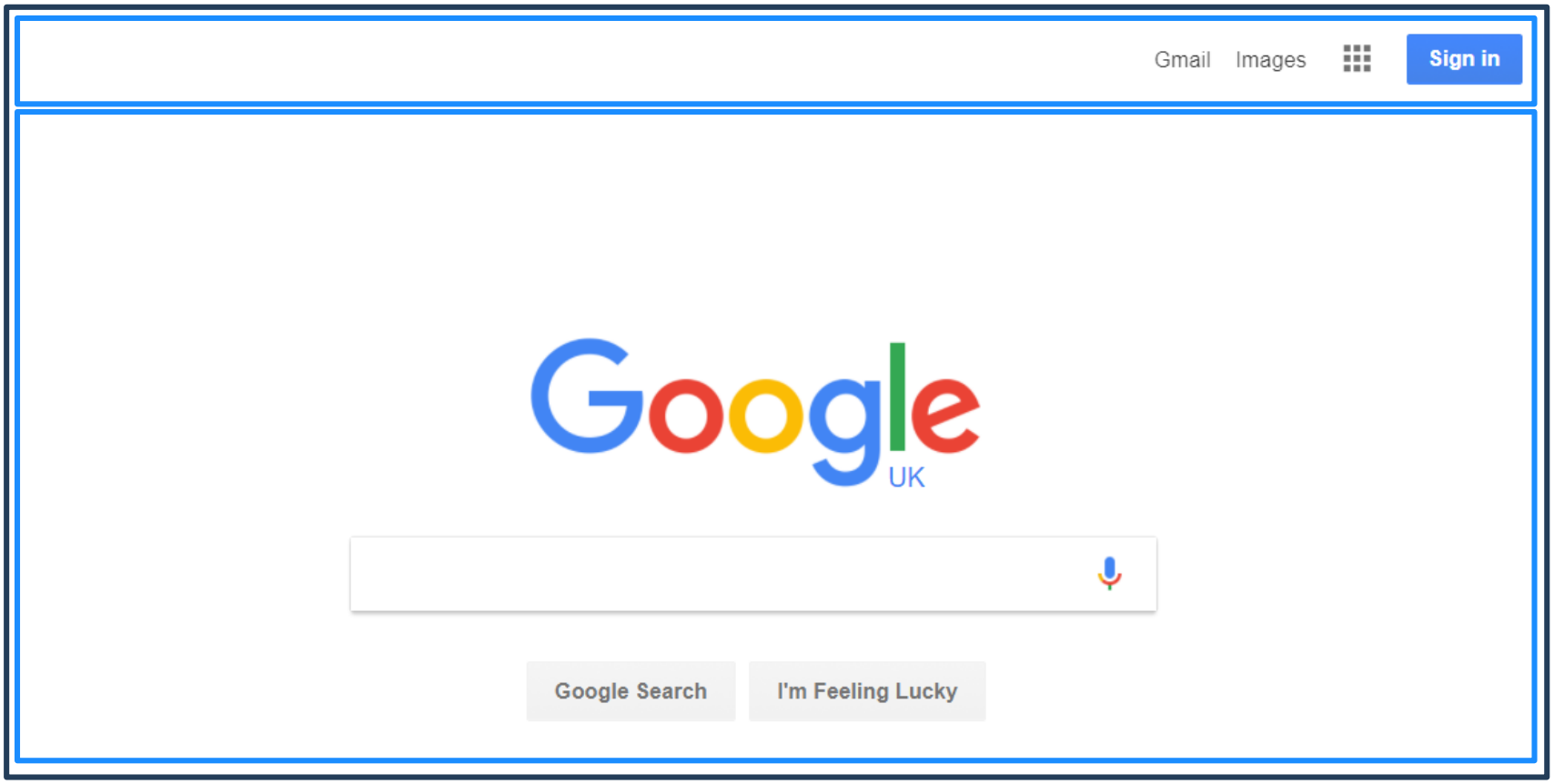
Boxifying Design



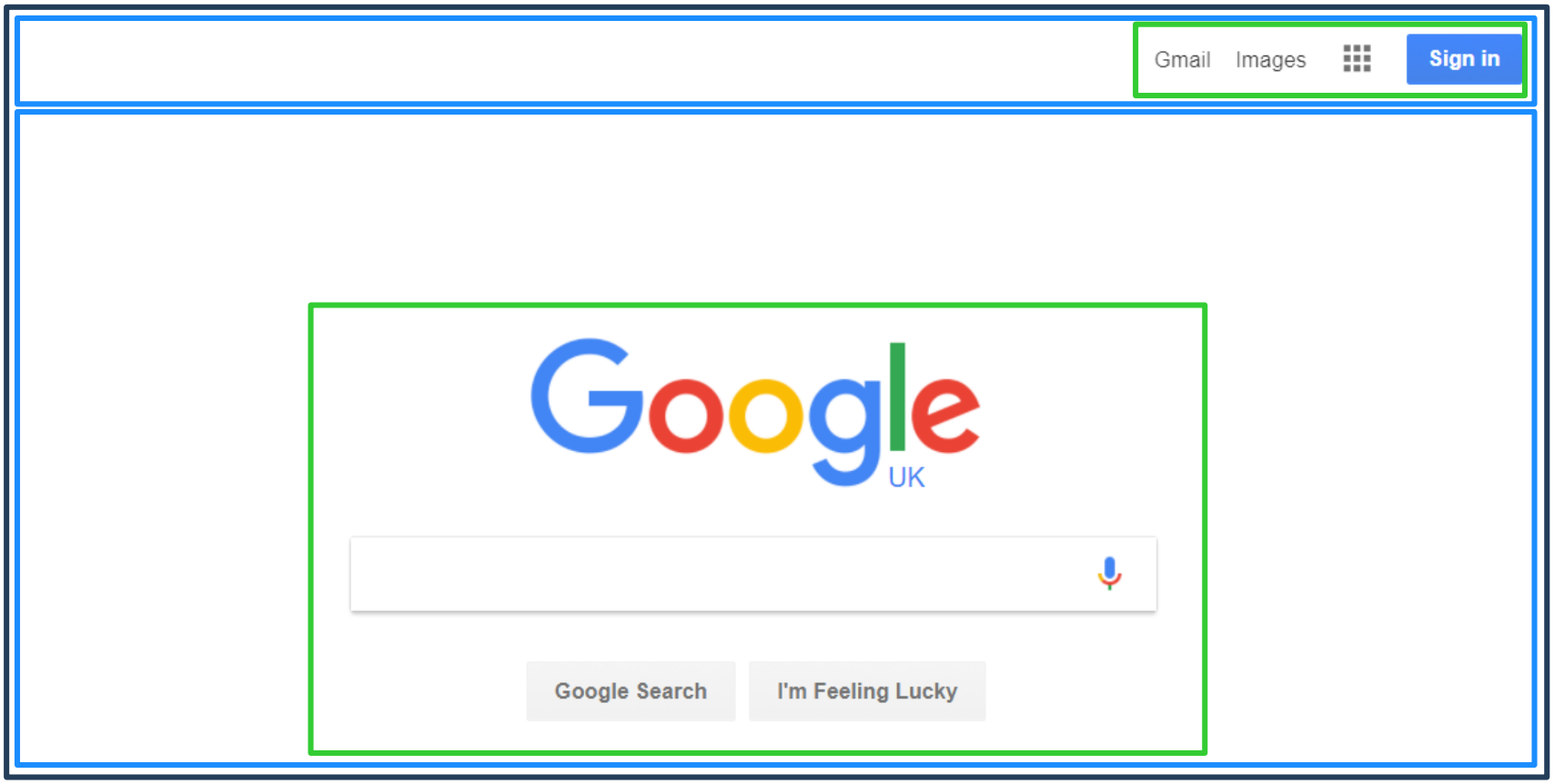
Boxifying Design



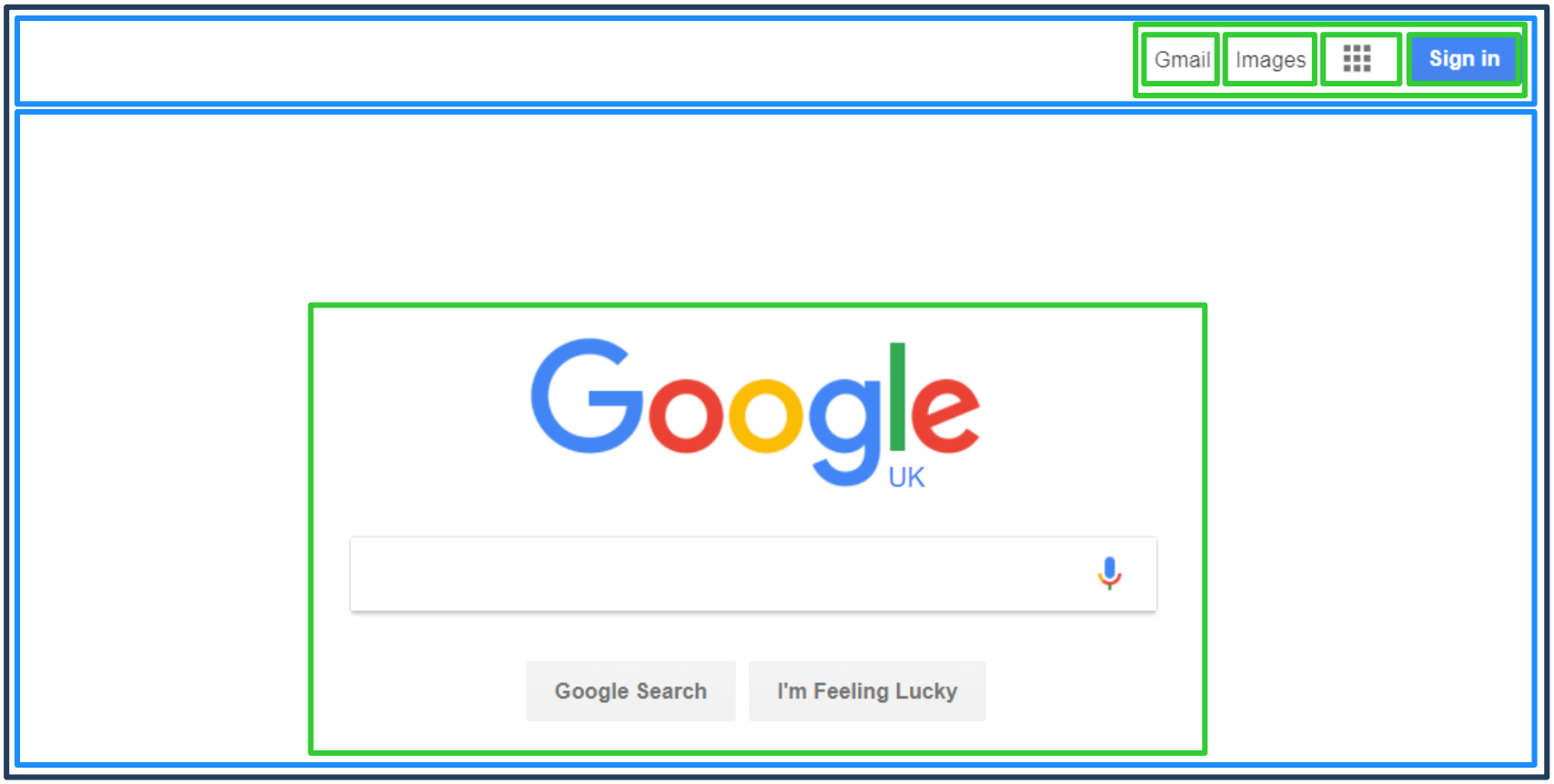
Boxifying Design



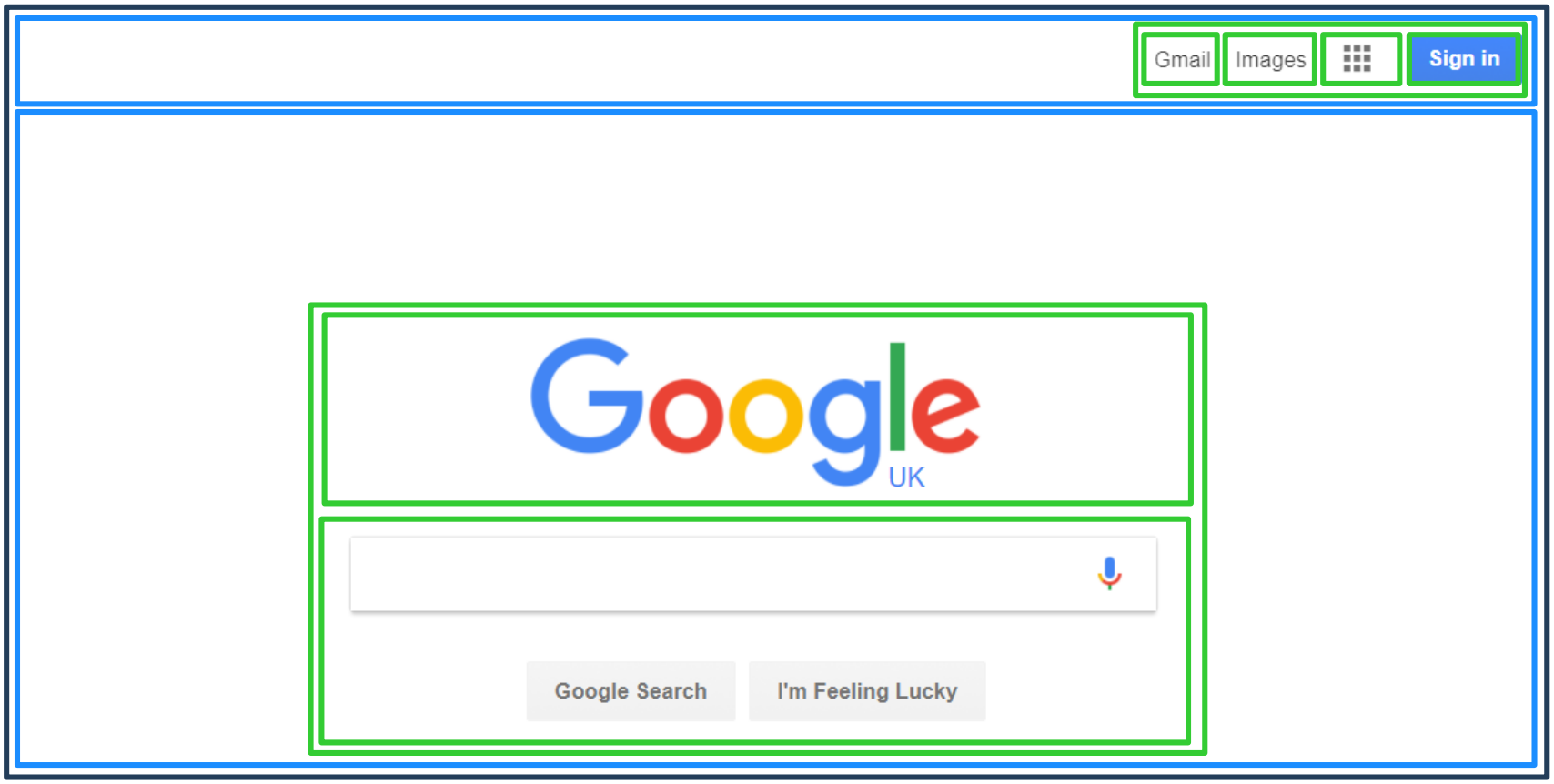
Boxifying Design



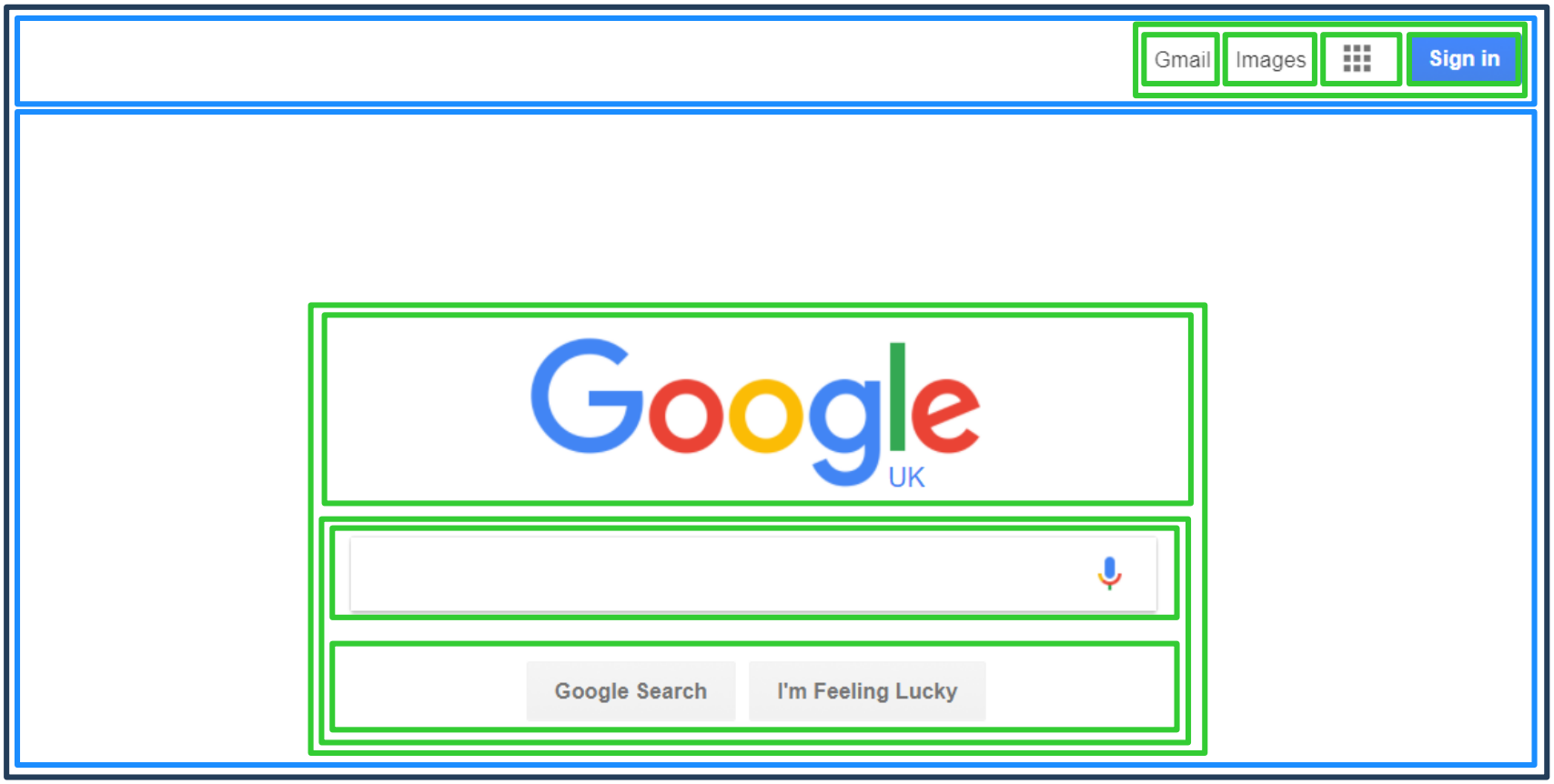
Boxifying Design



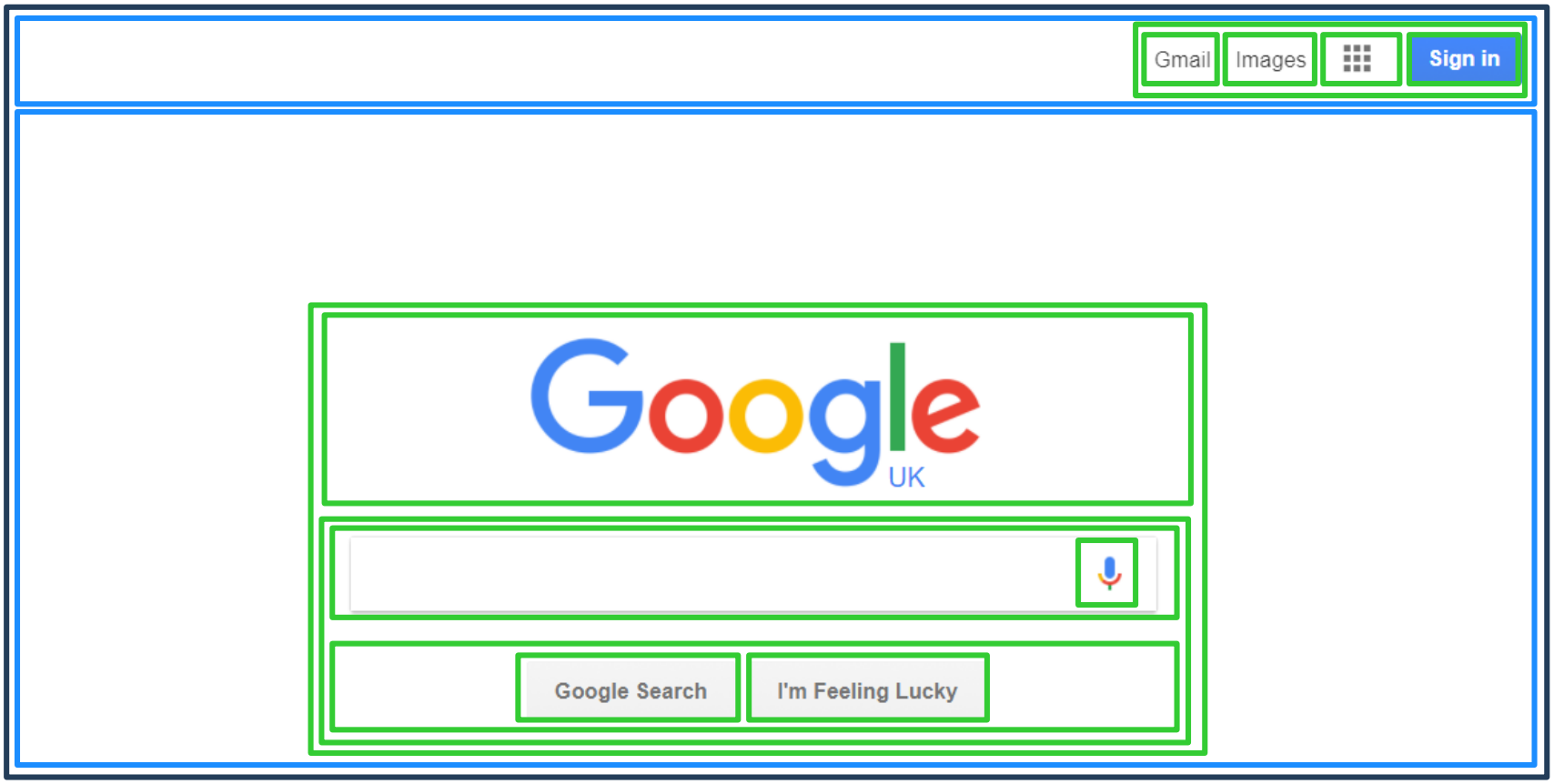
Boxifying Design



Boxifying Design

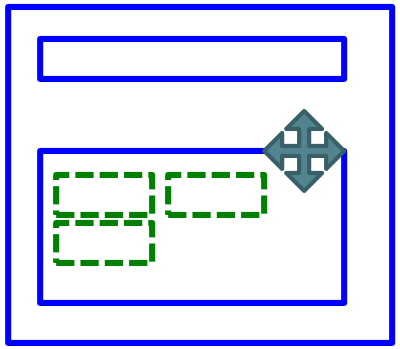


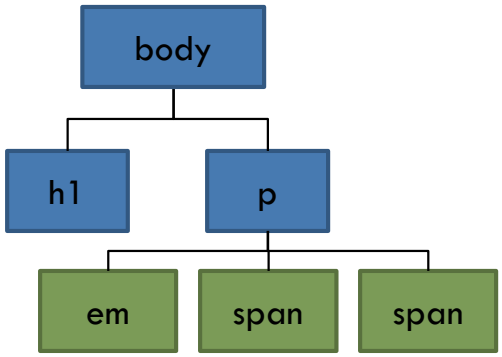
Boxifying Design



Displaying elements in document tree

- Typically, an element is rendered inside the box generated by an ancestor.
  - Exceptions: overflow and absolute positioning
- The exact position is determined by its positioning scheme.
  - You can fine-tune the position with some properties.





Positioning schemes

- CSS defines several positioning schemes
  - Normal flow (CSS1)
  - Floating (CSS1)
  - Absolute positioning (CSS2)
  - Flexible box (CSS3)
- We specify a positioning scheme for a box using two properties position and float.
  - The default values position: static and float: none select normal flow

Part A. Normal flow

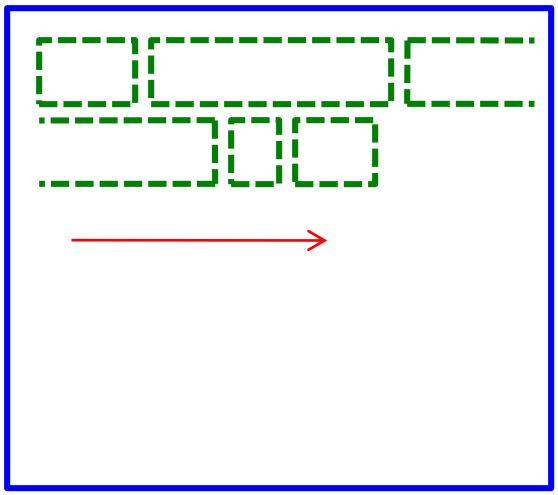
- Normal flow is the default positioning scheme
  - position: static; float: none;
  - Boxes arranged by normal flow are sometimes known as static boxes
- Normal flow arranges elements according to source order and box types
  - Inline boxes follow the flow of a line. They are laid out from left to right, line by line.
  - Block boxes are laid out vertically, from top to bottom.

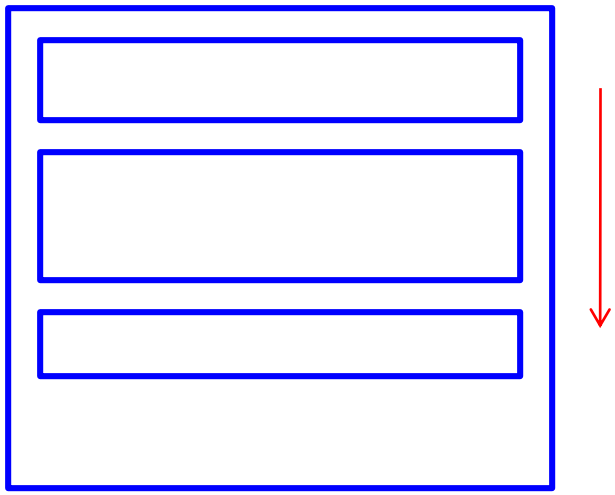
Inline box vs. Block box

Normal flow lays out inline and block boxes in different way.

- *Inline box*, generated by an element with display: inline
  - By default, inline elements (e.g. <a> <img> <input> <button>) generate inline boxes
- *Block box*, generated by an element with display: block
  - By default, block elements (e.g. <h1> <p>) generate block boxes
- You can change the box type of any elements
  - E.g. img { display: block; } lays out images in its own block

Boxes in Normal Flow





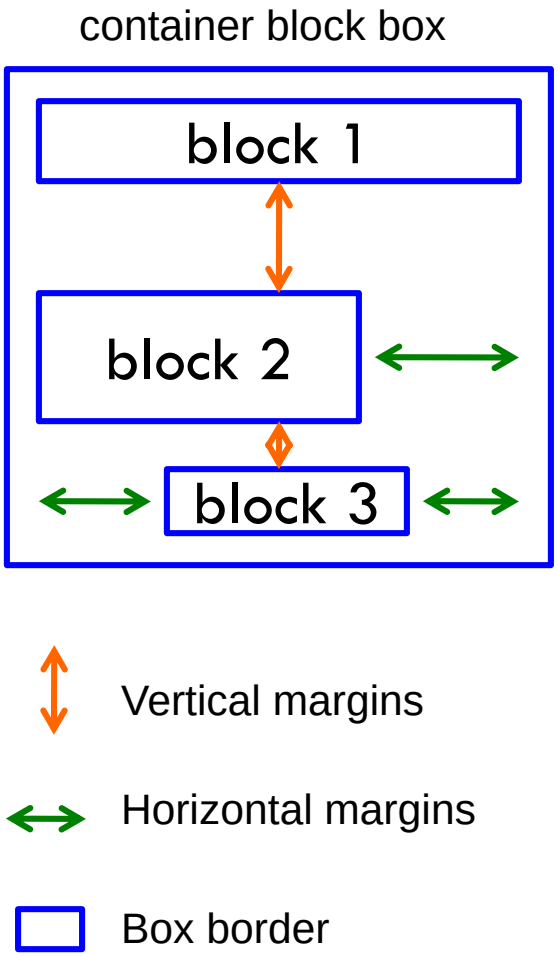
Inline boxes laid out inside a block box

Block boxes laid out inside a block box

Elements flows horizontally / vertically inside the nearest ancestor element that generates a block box.

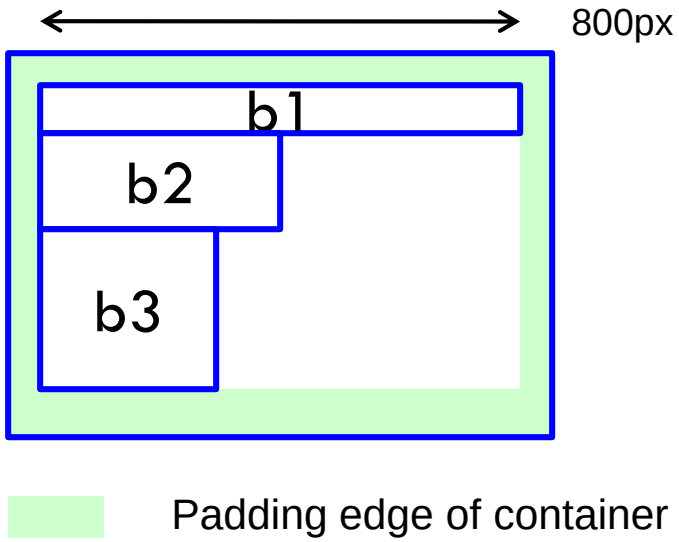
Normal flow of block boxes

- Block boxes are arranged vertically inside the container block, which is the block box generated by their parent element.
  - Fixed or relative width and height
  - Left and right margins adjust the horizontal position
  - Top and bottom margins adjust the vertical position



Width of block box

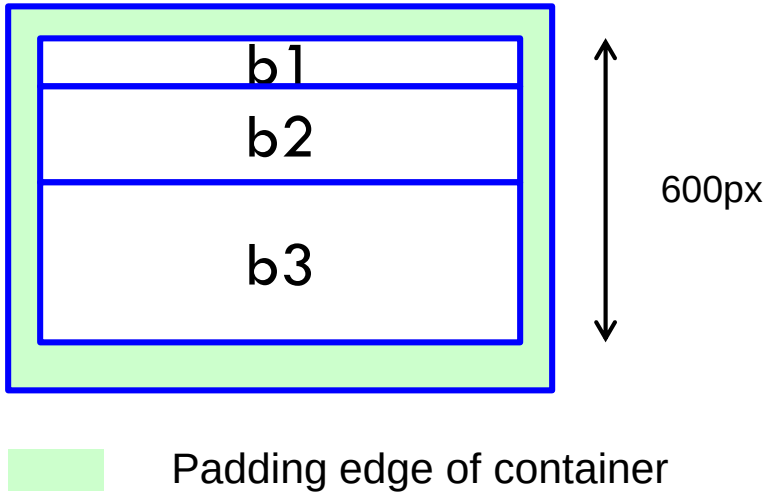

```
#container { width: 800px;
  padding: 30px; }
#b1 { width: auto; }
#b2 { width: 400px; }
#b3 { width: 30%; }
```



- Default width: auto stretches the width of the box to fill the container.
- Fixed value, e.g. width: 400px
- Percentage, e.g. width: 30%, calculated with respect to the width of the content area of the container

Height of block box


```
#container { height: 600px;
  padding: 30px; }
#b1 { height: auto; }
#b2 { height: 200px; }
#b3 { height: 50%; }
```



- Default height: auto makes the height large enough to contain the content of the box
- Fixed value, e.g. height: 200px
- Percentage, e.g. height: 50%, calculated with respect to the height of the content area of the container. (This is applicable only when the container height does not depend on its content.)

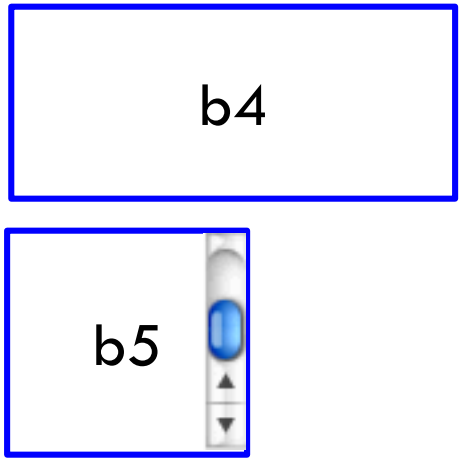
The overflow property

- If the box has fixed height, it may be too small to contain all its content. overflow specifies what should happen
  - overflow: visible (default) - overflow spills over the box.
  - overflow: hidden - overflow can not be seen.
  - overflow: scroll - the box scrolls to accommodate the overflow. Always show scroll bars.
  - overflow: auto - similar to scroll, but only show scroll bars when necessary.

```
#b4 { width: 800px; }

#b5 { width: 400px;
      height: 200px;
      overflow: auto; }
```

This example assumes that #b4 and #b5 have the same amount of text content.

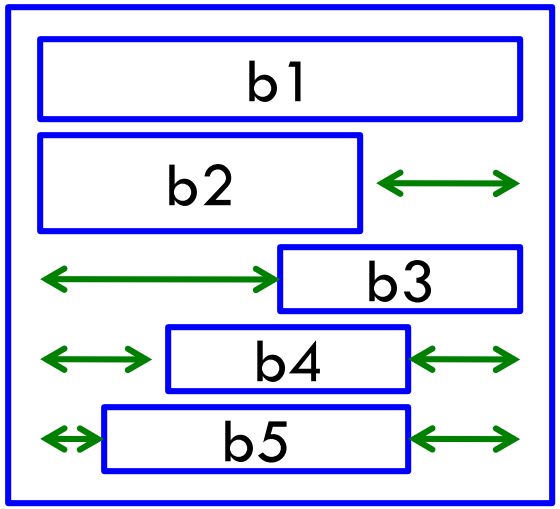


Constrain the dimension

- A box with width: auto or a percentage width changes its width when the browser window resizes
- Low readability when the box is too narrow or too wide
- The properties min-width and max-width set limit for the width
- The properties min-height and max-height set limits for the height

Left and Right Margins


```
#container { width: 800px; }
#b1 { width: auto }
#b2 { width: 500px;
      margin-right: auto; }
#b3 { width: 400px;
      margin-left: auto; }
#b4 { width: 400px;
      margin-left: auto;
      margin-right: auto; }
#b5 { margin-left: 100px;
      margin-right: 200px; }
```

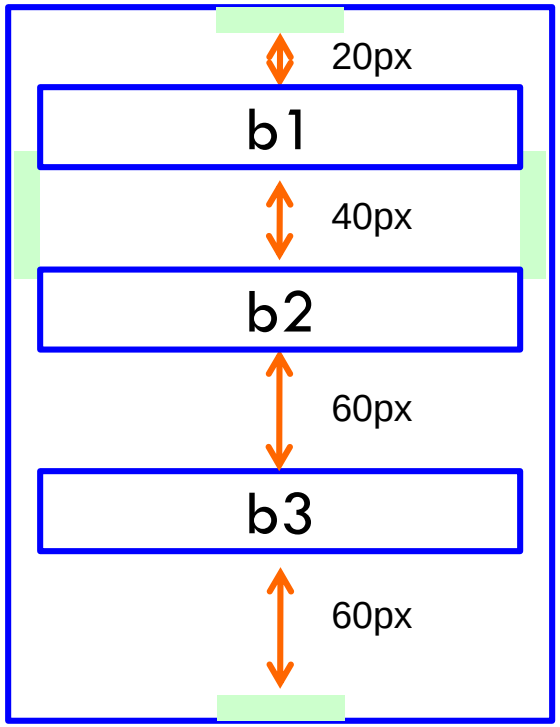


Container padding and vertical margins are added in this diagram for clarity.

By default, margin-left=0 and margin-right=0. If one of width, margin-left and margin-right is auto, it will extend to fill the container. If both margins are auto, the box will be centered horizontally.

Top and Bottom Margins


```
#container { padding: 10px; }
#b1 { margin: 20px 0; }
#b2 { margin: 40px 0; }
#b3 { margin: 60px 0; }
```



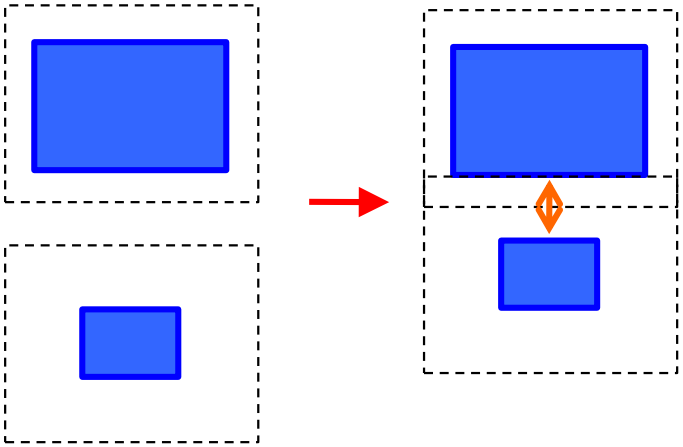
Padding of container

Vertical margins

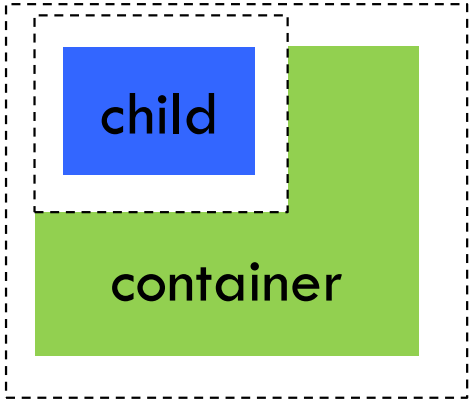
Vertical spacing between two adjacent boxes is the maximum of the bottom margin of the upper box and the top margin of the lower box.

Margin Collapsing

When two vertical margins touch, they collapse.



The bottom margin of an upper box and the top margin of a lower box collapse to the maximum of the two margins.

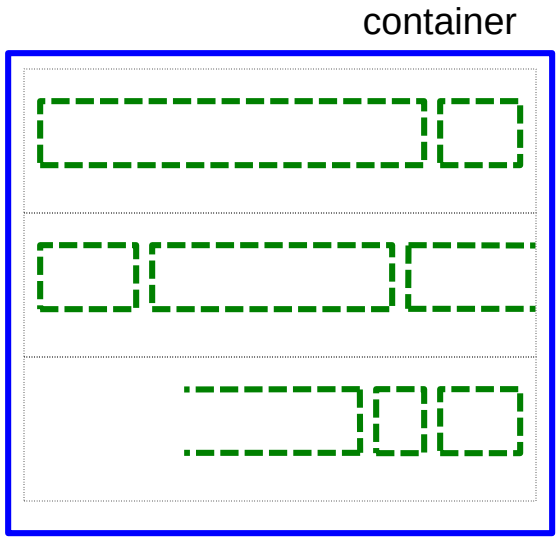


When a container block box <u>has no border and padding</u>, the top margin of a child box will touch its top margin. These two top margins will collapse also.

http://reference.sitepoint.com/css/collapsingmargins

Normal flow of inline boxes

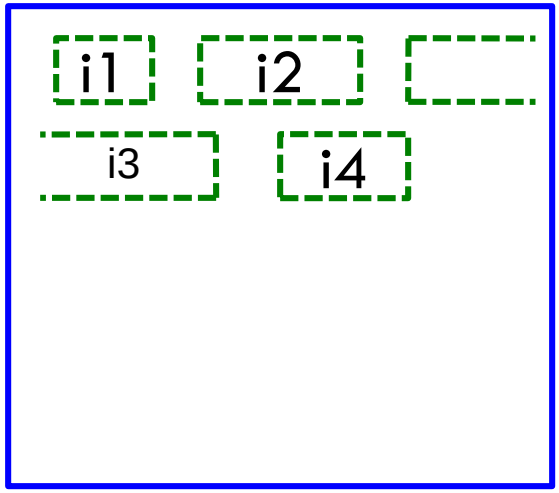

```
#container {
   display: block;
   line-height: 2;
   text-align: right;
 }
```



Inline boxes are laid out on line boxes stacked from top to bottom inside a block box. The property line-height gives the minimum height of the line boxes. Each line box should be tall enough to keep the content of inline boxes on the line.

The property text-align of the container also affects the positions of inline boxes in the line boxes.

Padding and Margins




```
#i1, #i2, #i3, #i4 {
    display: inline;
    margin-left: 10px;
    margin-right: 10px;
}
```

The horizontal padding and margins are observed when laying out inline boxes in lines. The vertical padding and vertical margins *do not* affect line height.

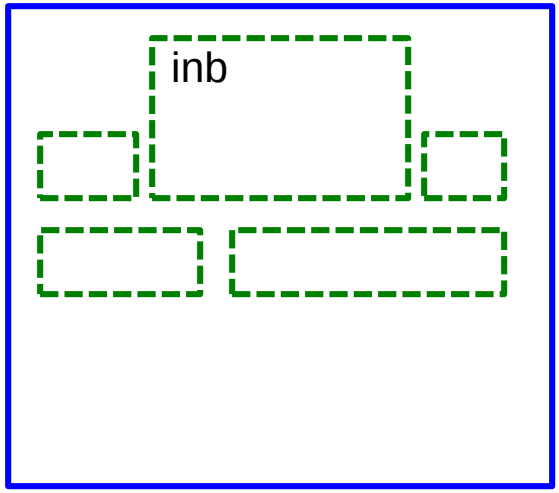
Width and height of inline boxes


```
<style>
  p#xy span { width: 100px; height: 2em; }
  p#xy img { width: 60px; height: 60px; }
</style>

<p id='xy'>
    Two inline boxes: a span <span>like this</span>
    and an image <img src='tick.png' alt=''/>.</p>
```

You can set the width and height of 'replaced elements' like <img> <iframe> and <object>. On the other hand, the browser ignores the width and height setting for other ('non-replaced') inline boxes. The browser fits the box size according to the content.

Inline-block box




```
#inb {
    display: inline-block;
    width: 100px;
    height: 50px; }
```

A special kind of inline box, specified with display: inline-block, behaves as if a block box is embedded inside an inline box. You can set the dimension of such inline-block box, and the line box will be tall enough to contain its margin and padding, not just its content.

Other box types

- CSS defines other box types
  - display: none – no box is generated. Normal flow ignores the element in layout
  - display: list-item – similar to block, but insert a bullet
  - display: table, and others – special box types for table, rows, columns, cells, etc

- visibility: hidden / visible only hides or shows an element. Normal flow still allocates space for the element.

vertical-align for text

vertical-align changes the vertical position of inline boxes in line box. The following values are common for text.

By default, text of **different size** and *different font* are aligned on baseline.

In addition, you can also align them as superscript like this and subscript like this.

value	meaning
baseline	(default) align this element baseline with the baseline of the default font of the line
super	Align this element as a superscript of the default font of the line
sub	Align this element as a subscript of the default font of the line

vertical-align for images

The following values are commonly used with image

value	meaning
bottom	Align the bottom of this element with the lowest element on this line
middle	Align this element in the middle of this line
top	Align the top of this element with the top of the highest element on this line

Here are several smileys  of different sizes 

Further reading

- vertical-align
  - http://css-tricks.com/what-is-vertical-align/
  - http://blog.themeforest.net/tutorials/vertical-centering-with-css/
  - http://www.w3schools.com/Css/pr_pos_vertical-align.asp
- Negative margin moves a box or its neighbors
  - http://www.smashingmagazine.com/2009/07/27/the-definitive-guide-to-using-negative-margins/
- Vertical alignment. Vertical centering
  - http://phrogz.net/css/vertical-align/index.html

Part B. Floating

- A float is a box that is shifted to the left or right of its container box
  - Selected by float: left or float: right
  - Inline content flows along a floated box
  - Floated boxes are laid out side by side

This diagram illustrates how text flows around a float.

The image is here (X) After the image, here are some more sentence. Notice how the image is taken away from the normal flow of inline boxes, and how the line boxes of content alongside a float are shortened to make room for the float box (including margins).

#b1 Landscape of Penha Hill in Macau

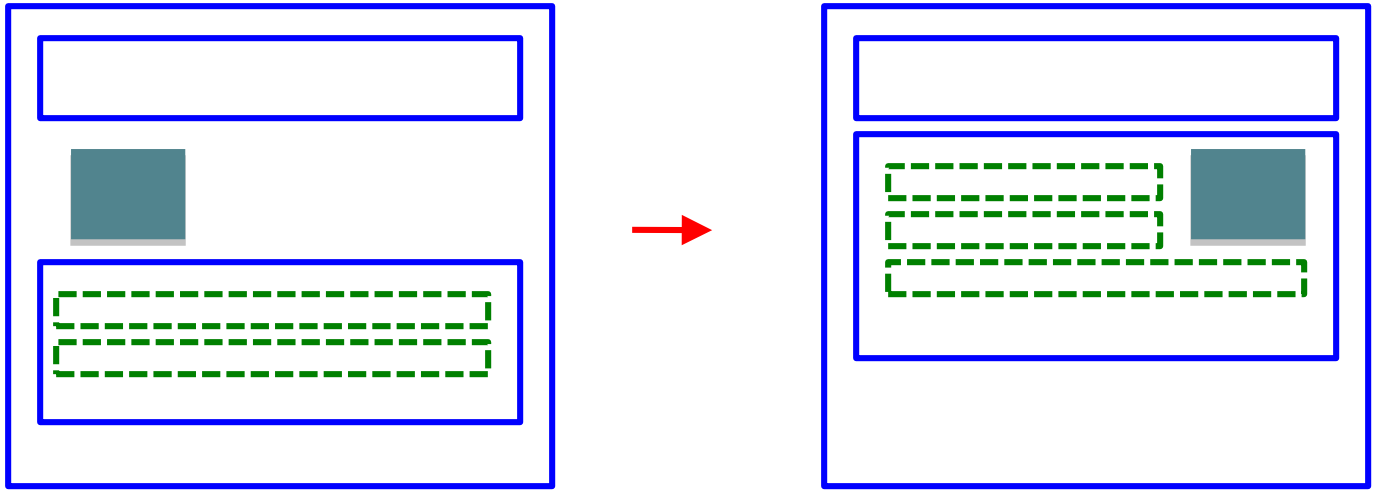
#b2 Macau viewed from Macau Museum

#b3 Casinos in Macau

Basic operation of floating

- Determine the ‘static position’ of the float, i.e. its position when laid out in normal flow
- Shift the float to left / right of the container box
- Remove the float from the flow
  - Elements after a float move up, as if the float does not exist
  - However, line boxes created after the float are shortened to make room for it
  - Do not affect preceding blocks

Floating and Normal Flow



Inline boxes in a paragraph

Block boxes for paragraphs

A floated block box

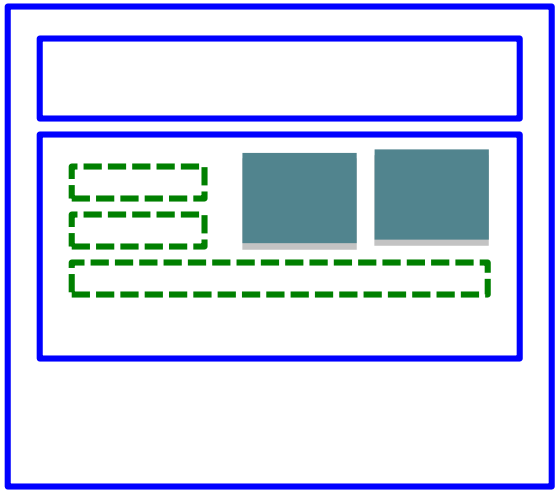
Block boxes are positioned as if the float does not exist, but inline boxes flow around the float.

More about floats

- You can float both inline and block boxes
  - When an inline box floats, it changes to block box
  - You should specify width of floats
- No margin collapsing on floats
  - between two floats
  - between a float and a static box

Multiple floats

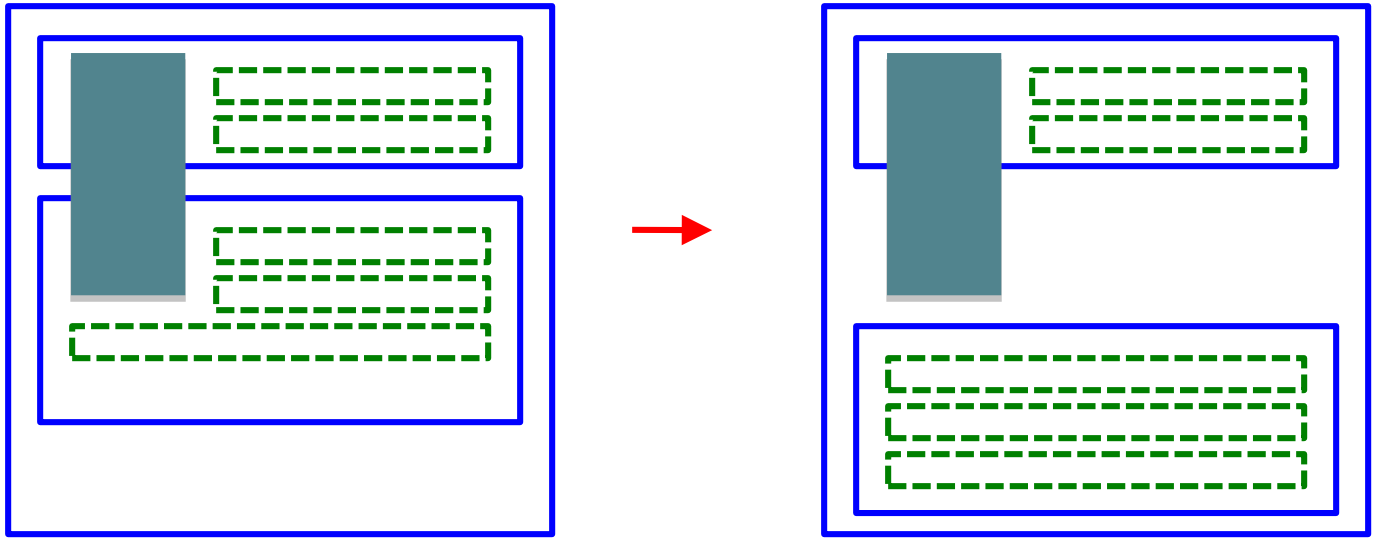
- When adjacent elements in HTML source code are floated to the same side, they are arranged side-by-side
  - If no enough space, a float goes to the next line below the previous float



Clearing from floats

- Clearing moves a block box down until it does not overlap floats
- The clear property
  - clear: left – clear on the left side
  - clear: right – clear on the right side
  - clear: both – clear on both side
  - clear: none (default)

Clearing from floats

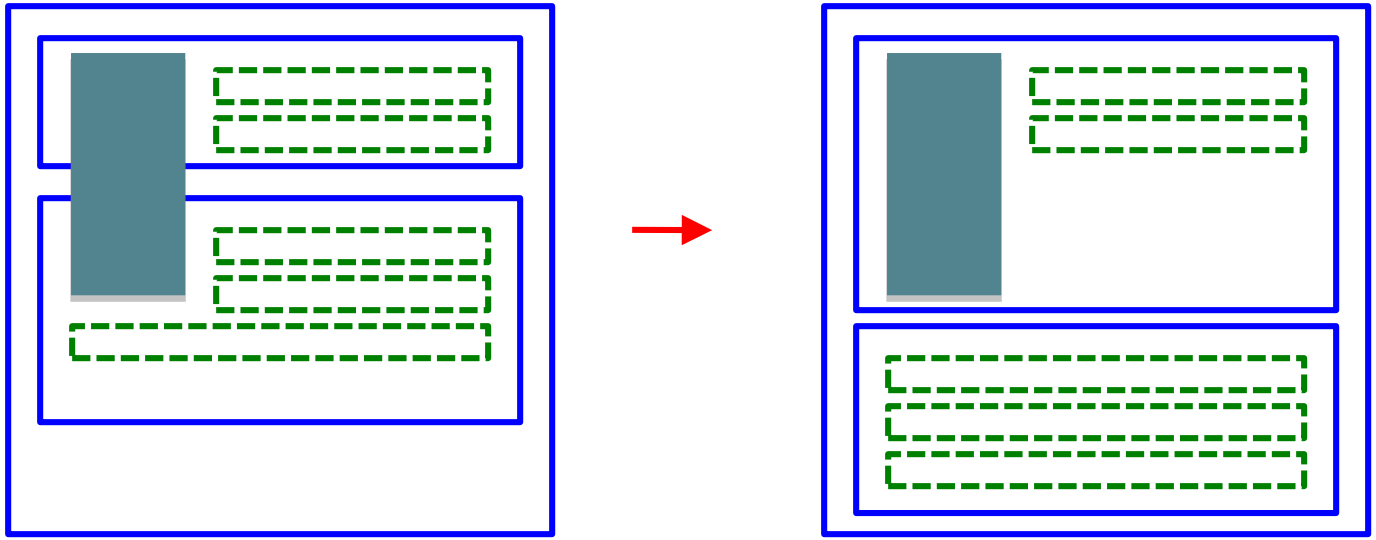


A float can affect line boxes of more than one block box. In this example, we use clear: left to clear the second paragraph from any floats on the left.

Clear the container

- The height of a container box with height: auto is only large enough to contain normal flow content inside it
  - Floats may overflow
- Two common methods to clear the container
  - Set overflow: auto for the container
  - Float the container (note: you can float a box inside another float)
  - Ref. http://blogs.sitepoint.com/2005/02/26/simple-clearing-of-floats/

Clearing the container



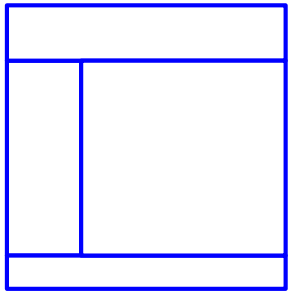
Clearing the container prevents a descendant float to affect other boxes.

Laboratory

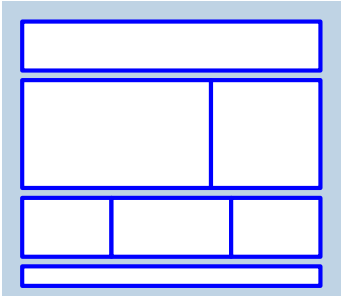
- Lab 5-3. Basics of floating
  - Floating inline content
  - Floating block boxes. Clearing.
- Lab 5-4. Floating for horizontal layout
- Lab 5-5. HTML form formatting by floating
  - No <table>
  - Left / right alignment of labels

Part C. Web page layout

- Modern web pages usually divide their content into boxes and arrange them in rows and columns.
- HTML markup for boxes
  -
  - New elements in HTML5
- Implementing layout
  - Floating
  - Fixed width vs. liquid width



Column layout

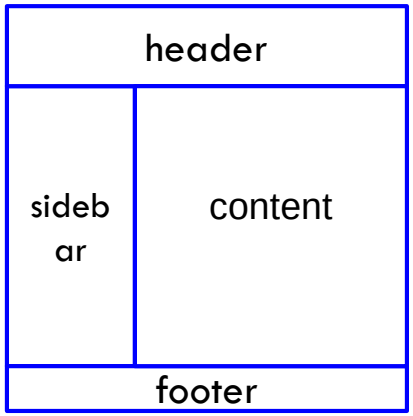


Grid layout

Using

- It is common to use for the boxes in page layout
- Describe the function of a box with the id or class attribute

```
<body>
  <div id="header">..</div>
  <div id="sidebar">..</div>
  <div id="content">..</div>
  <div id="footer">..</div>
</body>
```



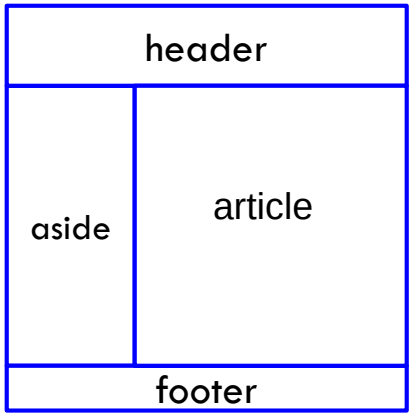
New elements in HTML5

- HTML5 defines some elements for common content in web pages
  - <header> – a header usually contains logo, title, and navigation menu
  - <footer> – a footer usually contains the author, copyright data, related links, and contact info
  - <aside> – consists of info that is tangentially related to the content around. Usually arranged as sidebar
  - <nav> – a navigation consists of links to other doc or parts in this doc
  - <section> – consists of some related content, typically with a heading. Examples include chapters and sections in a book.
  - <article> – a composition that forms an independent part of a page, e.g. forum post, magazine or newspaper article

Example

- This example assumes that the main content of the page is a newspaper article, and the sidebar provides some related info about the article.

```
<body>
  <header>..</header>
  <aside>..</aside>
  <article>..</article>
  <footer>..</footer>
</body>
```

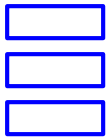


Example


Typo
<header>
<nav>
1. Home
2. About
3. Archives
4. Contact
Getting your stock photos seen
<article>
Top 10 tips for new bloggers
<article>
10 fantastic photography tips
<article>
« Older posts
<nav>
Newer posts »
<aside>
About me
<section>
Categories
Social
<section>
Latest posts
<section>
Search
<section>
<footer>

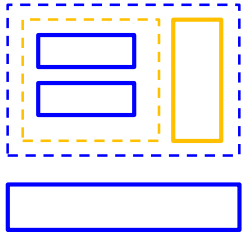
Basic strategy in CSS layout

- Normal flow arranges block boxes vertically



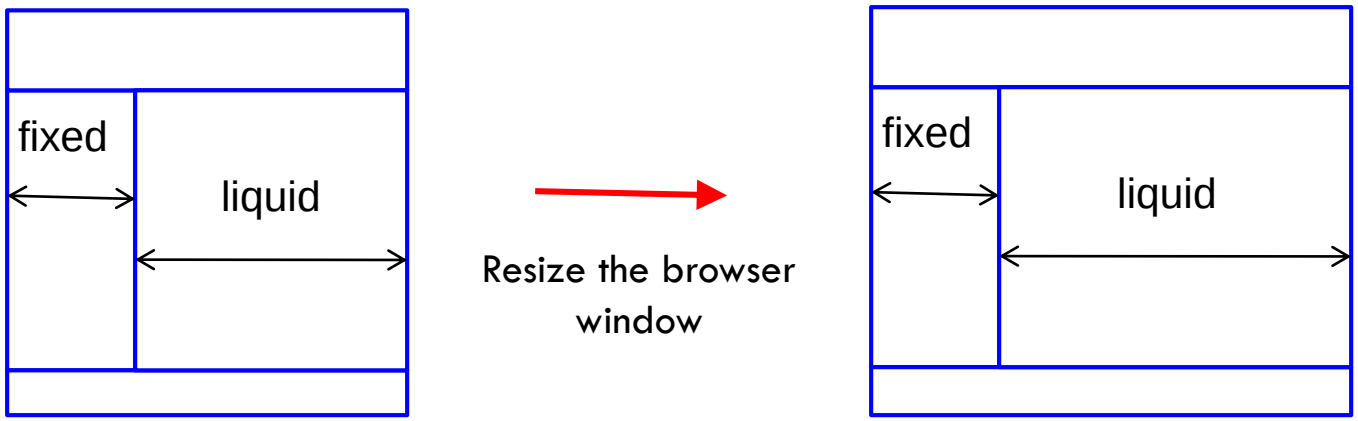
- Floating arranges block boxes horizontally in a row.

- You can nest appropriately to mix the layout direction



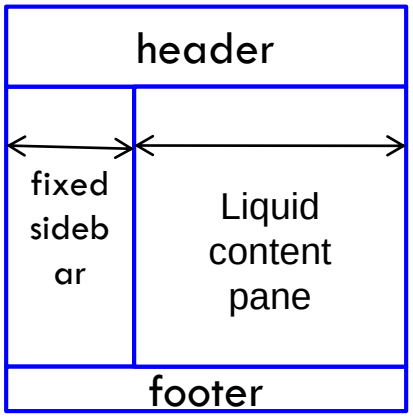
Liquid vs. fixed width

- A box with liquid width fills the remainder of the browser window
  - Liquid width also known as flexible, fluid or elastic.



Column layout

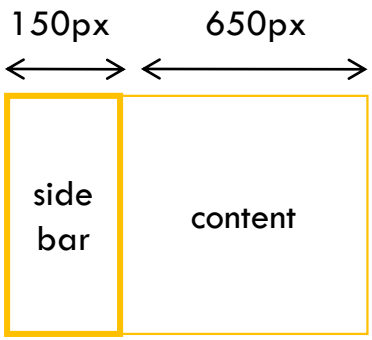
- We will use CSS to make a two-column layout with the following boxes
  - header contains logo
  - A sidebar div#sidebar has fixed width
  - Content pane div#content has liquid width. It is usually taller than the sidebar
  - footer contains copyright info



Float both columns

- One way to arrange the columns in a row is to float all columns
- Shortcoming: you must fix the width of both columns. Non-flexible layout

```
#sidebar {
  float: left;
  width: 150px;
}
#content {
  float: left;
  width: 650px;
}
```

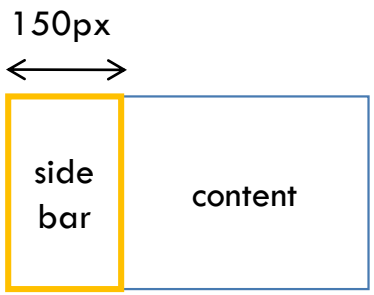



```
<body>
  <div id="sidebar">..</div>
  <div id="content">..</div>
</body>
```

Float one column

- Another method is to float the sidebar only
- The content pane uses the default width: auto.
- Benefit: liquid width
- To prevent content of the content pane to wrap around the sidebar , add a left margin to the content pane.

```
#sidebar {
  float: left;
  width: 150px;
}
#content {
  margin-left: 150px;
}
```



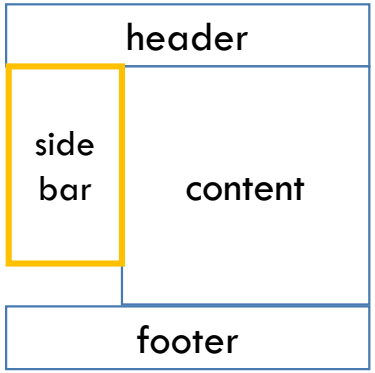

```
<body>
  <div id="sidebar">..</div>
  <div id="content">..</div>
</body>
```

Two columns with header & footer

- Header, content and footer follow normal flow
- The floated sidebar is under the header because it floats to the left from its static position

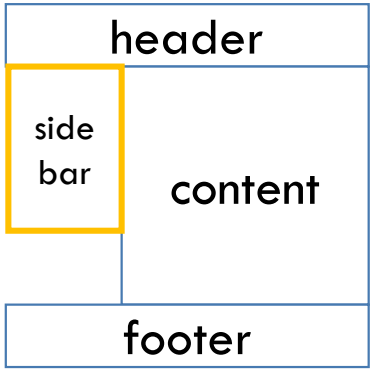
```
#sidebar { float: left; width: 150px; }
#content { margin-left: 150px; }
```

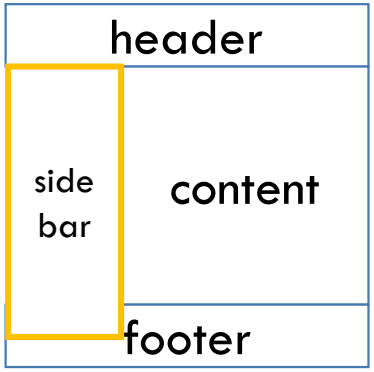
```
<body>
  <header> ... </header>
  <div id="sidebar"> … </div>
  <div id="content"> … </div>
  <footer> … </footer>
</body>
```



Different lengths of columns

- Unless you fix the same height for both columns, they may have different height
  - You may paint the container background color appropriately to hide that fact that the sidebar that is too short.
  - If the sidebar is taller than content, it will overlap the footer.



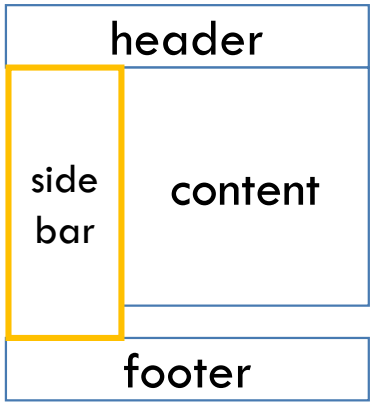


Floating the sidebar

- A solution is to clear the footer from floats
  - Footer is always below content pane because of normal flow

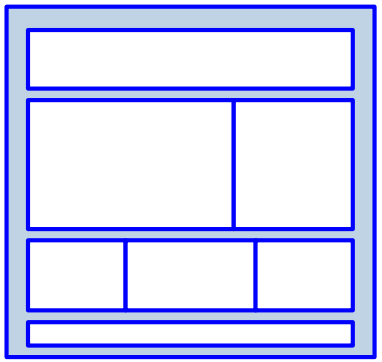
```
#sidebar { float: left; width: 150px; }
#content { margin-left: 150px; }

footer { clear: left; }
```

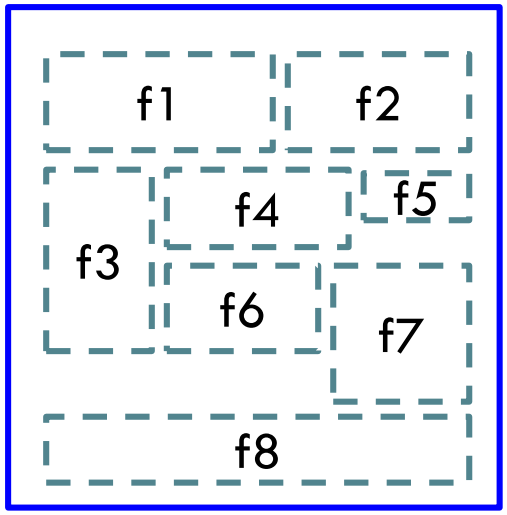


Grid layout

- The web page is divided into several rows of the same width but different height
- Some row may be further divided into columns
  - We can use floating to position the boxes
  - CSS frameworks provide a convenient way to implement such layout.
    - Blueprint, http://www.blueprintcss.org/
    - 960 grid system, http://960.gs/



Floating multiple boxes




Floating box

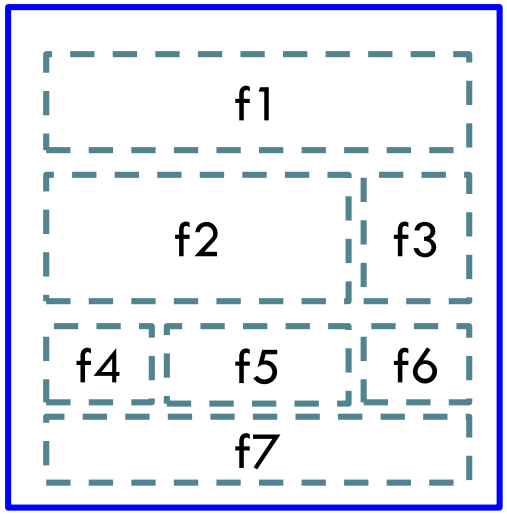

Container block box

```
div.container div  { float: left; }

<div class="container">
   <div id="f1"> … </div>
   <div id="f2"> … </div>
   …
   <div id="f8"> … </div>
</div>
```

When there is enough room, a float is put beside a previous one (e.g. f2). Otherwise, it may go to the beginning of the next row (e.g. f3) or under a previous one (e.g. f6).

Grid layout using floats

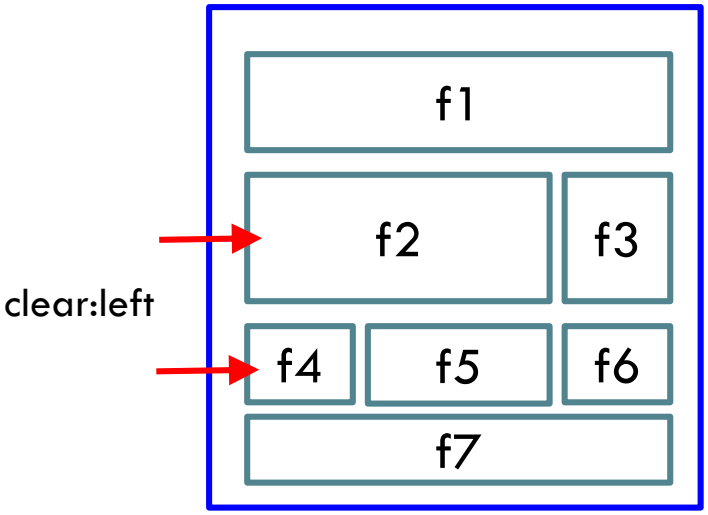



```
div.container div  { float: left; }
div.container { width: 800px; }
#f1 { width: 800px; }
#f2 { width: 600px; }
#f3 { width: 200px; }
#f4 { width: 200px; }
…
```

A simple strategy to make a grid layout is to fix the widths of all boxes and float them. Make sure the total width of boxes on a row is the same as the container width.

Pay attention to padding, border and margins of the boxes. They increase the total width required on a row.

Clear to open a new row




```
div.container div  { float: left; }
div.container { width: 800px; }
#f1 { width: 800px; }
#f2 { width: 600px; clear: left; }
#f3 { width: 200px; }
#f4 { width: 200px; clear: left; }
…
```

To make sure a box goes to a new row, either enforce the same height for all boxes on the previous row, or set **clear:left** for the box (e.g. f4). This will move the box below all floats before it.

Web page layout

Chapter 5, part 2

Part D. Absolute positioning

- Absolute positioning places a box in a specified location on a layer above the containing block
  - Selected by position: absolute
  - Offset properties left, right, top and bottom determine the location of the box
  - Property z-index determines the stacking order of layers
- Elements using absolute positioning become block boxes
  - commonly known as AP boxes or AP layers.
  - Should specify width. May specify other box properties

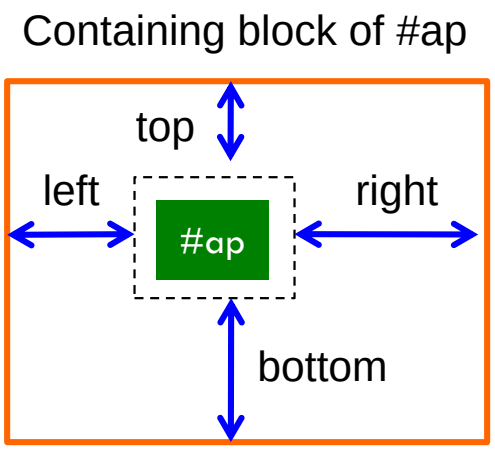
Basic operation of abs pos

- Remove the AP box from the flow
  - Normal flow sibling elements of the AP box remain in the containing block *beneath* the AP box
  - The AP box *does not affect* the layout of these content in the containing block
- Use offset properties to determine the location of the AP box w.r.t. the containing block.
  - If no offset are specified, use the 'static position', i.e. its position when laid out in normal flow

Offset properties

- You can set location of the AP box by specifying offset from two sides:
  - If left / right has a value, it is used to fix the horizontal position.
  - If top / bottom has a value, it is used to fix the vertical position.
  - If both left and right have a value, and width: auto, then the browser stretches the box horizontally to fill the containing block
    - Similar for top and bottom.

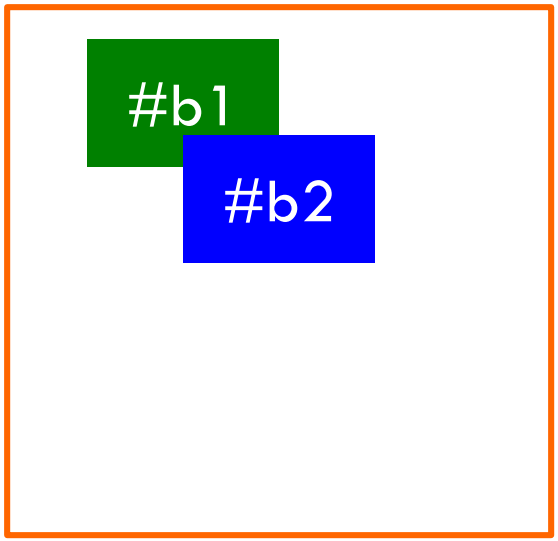
```
#ap {
  position: absolute;
  top: 20px; right: 20px;
}
```



z-index property

- AP box may cover other AP boxes
- The property z-index determines the stacking order
  - Box with larger z-index is on top of the other box

```
#b1, #b2 {
   position: absolute;
   width: 100px;  height: 80px; }
#b1 {
   background-color: green;
   top: 10px; left: 30px;  z-index: 2; }
#b2 {
   background-color: blue;
   top: 70px; left: 80px;  z-index: 6; }
```

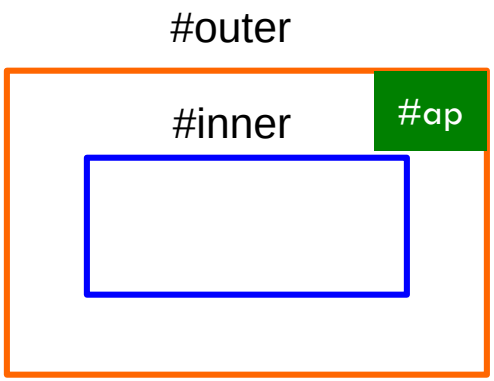


Containing block

- The *containing block* for an AP box is the nearest positioned ancestor.
  - A positioned element is one with position: absolute, position: relative or position: fixed
  - Boxes using normal flow and floating are not positioned
  - Containing block includes the padding edge
- If there is no such ancestor, CSS will use the 'initial containing block', which is the region of the HTML document within the initial viewport of the browser

Containing block, example

- The *containing block* for an AP box is the nearest positioned ancestor.



#outer is the containing block of #ap.

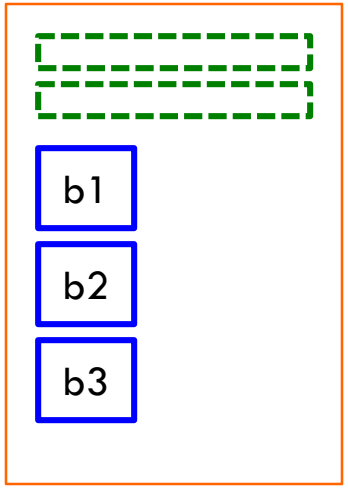
```
#outer { position: relative; }
#inner { position: static; }
#ap { position: absolute;
  top: 0; right: 0;
}

<body id="outer">
   <div id="inner">
        <div id="ap">...</div>
   </div>
</body>
```

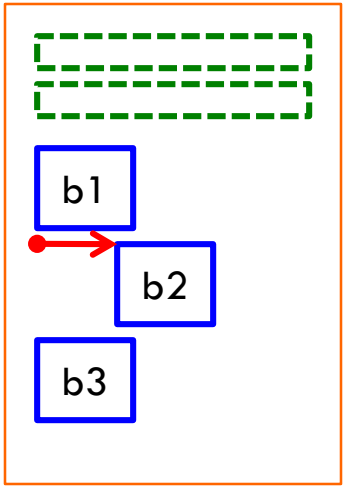
Relative positioning

- Relative positioning offsets a box from its original position in normal flow
  - Selected by position: relative
  - The browser first lays out the box in normal flow, then offsets the box from this 'static position' according to properties top, bottom, left, right
    - E.g. "top:10px" moves the element *down* 10px
  - Surrounding elements are laid out as if the box is still in normal flow.
  - Useful in turning a parent into "positioned" when laying out children using absolute positioning

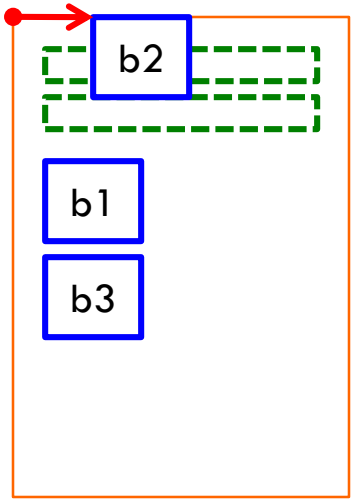
Relative vs. Absolute Positioning



position: static



position: relative;
left: 20px; top: 0px;



position: absolute;
left: 20px; top: 0px;

Block box of body
(assume position:relative)

Block boxes in body

Relative and absolute positioning use different reference points in placing the box. In addition, abs pos removes the element from normal flow completely.

position: fixed

- This is a special kind of absolute positioning that uses the *browser viewport* as the containing block
  - Selected by position: fixed
  - Similar to background-attachment: fixed, such box does not scroll with the content of the page
    - E.g. "bottom: 0" means that the bottom of this box touches the bottom of the viewport.

Summary of positioning schemes

Scheme	How to choose	Box type	Positioning
Normal flow	position: static	inline	left to right, line by line
		block	top down
Floating	float: left / right	block	Vertical position as in normal flow. Push to a side.
Relative positioning	position: relative	inline / block	Offset from position in normal flow (top/right/bottom/left)
Absolute positioning	position: absolute	block	top/bottom sets the vertical position. left/right sets the horizontal position. These properties refer to the nearest positioned ancestor.
	position: fixed	block	Same as position: absolute, except positioning refers to browser viewport.

Part E. Flexible box model

- CSS3 introduces the flexible box model
  - Distribute boxes horizontally / vertically inside a container box
  - Boxes with flexible widths share remaining spaces in the container box
  - The boxes may be arranged in an order different from source order
  - Boxes in a row may stretch to the same height (similar to table cells in a table row)

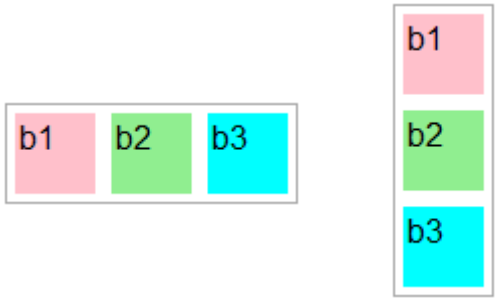
Flexbox is still a working draft

- The flexible box layout module is still a working draft
  - Not a standard yet
  - Firefox and WebKit based browsers (Safari, Chrome) have experimental implementation
  - Need to add a prefix
    - -moz- Mozilla Firefox
    - -webkit- Apple Safari and Google Chrome

Using flexible box model

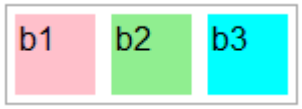
- Use display: box to select flexible box model for a block element.
- Children in the block element will be arranged either horizontally or vertically
  - box-orient: horizontal (default)
  - box-orient: vertical

```
<div id="C">
  <div id="b1"> … </div>
  <div id="b2"> … </div>
  <div id="b3"> … </div>
</div>
```

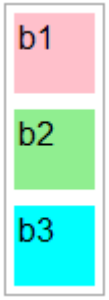


Example


```
#C {
 display: -moz-box;
 -moz-box-orient: horizontal;
}
#C div { width: 40px; height: 40px; }
```




```
#C {
 display: -moz-box;
 -moz-box-orient: vertical;
}
#C div { width: 40px; height: 40px; }
```



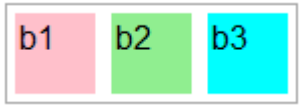
Layout order

- You can reverse the order that children are laid out with box-direction: reverse
- You can change the order that children are laid out by the property box-ordinal-group
  - The box model will distribute boxes with box-ordinal-group: 1 (default) first, then those with box-ordinal-group: 2, and so on.

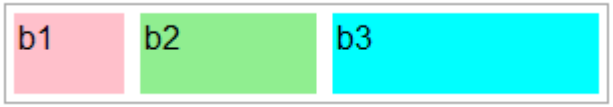
Flexible box size

- By default, a box is not flexible, and you can fix its width (e.g. width: 100px)
- A box becomes flexible when its property box-flex is at least 1
  - #b1 { box-flex: 1 }
- If some children have fixed width,
  - the box flexible model first allocates space in the container parent to them
  - the remaining space is then allocated to each flexible child box in proportion to its box-flex value.

Example




```
#C { display: -moz-box; width: 600px; }
#b1, #b2, #b3 { -moz-box-flex: 1; }
```



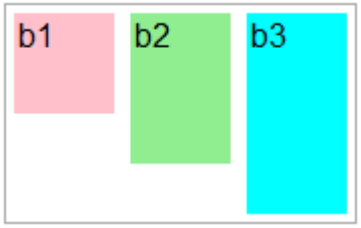

```
#C { display: -moz-box; width: 300px; }
#b1 { -moz-box-flex: 1; }
#b2 { -moz-box-flex: 2; }
#b3 { -moz-box-flex: 1;  width: 100px; }
```

Distributing remaining vertical spaces

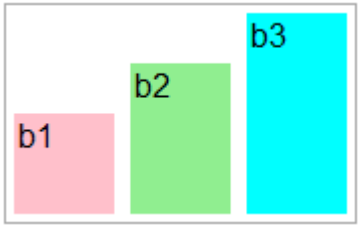
- (assuming box-orient: horizontal)
- Height of boxes may be different and smaller than parent's height. How the vertical space is distributed on the vertical axis is determined by box-align

box-align: start	align at the top of the parent
box-align: end	align at the bottom of the parent
box-align: center	each box is placed at the center
box-align: stretch	height of each box is adjusted to fit the height of the parent

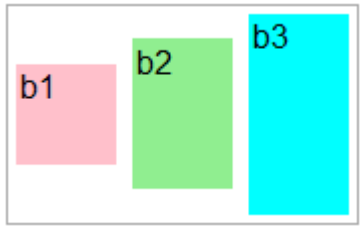
Example



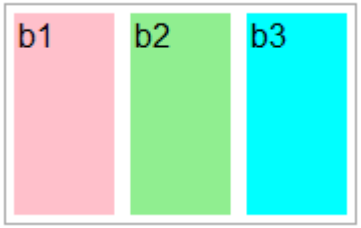
#C { -moz-box-align: start; }



#C { -moz-box-align: end; }



#C { -moz-box-align: center; }



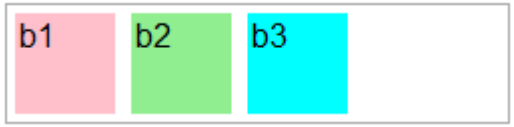
#C { -moz-box-align: stretch; }

Distributing remaining horizontal spaces

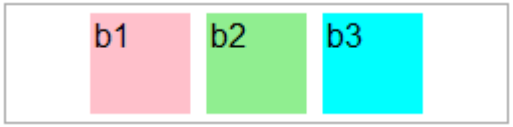
- (assuming box-orient: horizontal)
- The total width of child boxes may be smaller than the parent's width. How the remaining horizontal space is distributed is determined by box-pack

box-pack: start	child boxes are pushed to the left
box-pack: end	child boxes are pushed to the right
box-pack: center	child boxes are placed at center

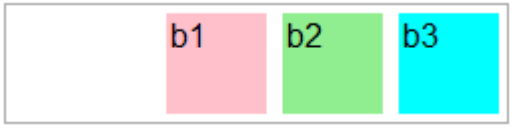
Example



#C { -moz-box-pack: start; }



#C { -moz-box-pack: center; }



#C { -moz-box-pack: end; }

Further readings

- A wiki about CSS: http://css-discuss.incutio.com/wiki/Main_Page
- CSS codes and library: http://www.dynamicdrive.com/style/
- Layout:
  - Layout gallery: http://blog.html.it/layoutgala/
  - Layout tutorial: http://www.maxdesign.com.au/presentation/page_layouts/
  - The new layout in DW CS5 http://www.adobe.com/devnet/dreamweaver/articles/introducing_new_css_layouts.html